Delft Object-oriented Radar Interferometric Software
User's manual and technical documentation

Version: v4.02

Delft Institute of Earth Observation and Space Systems (DEOS)
Delft University of Technology

TU Delft

Preface

This document describes the Doris Software for Interferometric SAR processing. It is compliant with Doris **v4.02**. This manual contains technical documentation and information that is required for running the software. We try to be as complete as possible, but some chapters may be very brief in the description. Please report any incompleteness in the documentation or the source code. The latest information on the Doris software can always be found on the internet: `http://enterprise.lr.tudelft.nl/doris/`.

Doris is freely available to the scientific community. The conditions of use for the Doris software are as follows.

1. Doris is a scientific-purpose software and cannot be commercialized, nor can parts or products of it be commercialized. Parties interested in using Doris or its products for any commercial purposes are requested to contact Prof.Dr. Ramon Hanssen of DEOS (r.f.hanssen@tudelft.nl)
2. Our version of the software is the only official one. Please do not distribute the Doris software to third parties, instead refer to the Doris home page. This in order to guarantee uniformity in the distribution of updates and information.
3. Delft University of Technology is not responsible for damage of any kind caused by errors in the software or in the documentation.
4. Users are very welcome to extend the capabilities of the Doris software by implementing new algorithms or improving the existing ones. It is intended that if new software is developed based on Doris, that this also is made available for free to the other users (through us).
5. We would appreciate if any addition or modification of the software would be announced first to us, so that it can be included in the official (next) version of the software.
6. Publications that contain results produced by the Doris software should contain an acknowledgment. (For example: The interferometric processing was performed using the freely available Doris software package developed by the Delft Institute of Earth Observation and Space Systems (DEOS), Delft University of Technology. or include a reference to: Bert M Kampes, Ramon F Hanssen, and Zbigniew Perski. Radar interferometry with public domain tools. In Third International Workshop on ERS SAR Interferometry, 'FRINGE03', Frascati, Italy, 1-5 Dec 2003, page 6 pp., 2003.

This document is typeset in LaTeX2e.

Delft, December 2008.

Contents

Preface **ii**

1 Introduction **2**
1.1 Overview of the InSAR Processing . . . 2
1.1.1 Processing order . . . 4
1.2 General considerations and conventions . . . 5
1.2.1 Inputfile . . . 6
1.2.2 Outputfiles . . . 6
1.3 Outline of this document . . . 8

2 General Cards **10**
2.1 General Input Cards . . . 10
2.2 Example General Input Cards . . . 12

3 M_READFILES **14**
3.1 Input Cards . . . 14
3.2 Output Description . . . 15
3.3 Implementation . . . 16
3.3.1 Changes for X86 platforms . . . 16

4 M_PORBITS **17**
4.1 Input Cards . . . 17
4.2 Output Description . . . 18
4.3 Implementation . . . 19

5 M_CROP **20**
5.1 Input Cards . . . 20
5.2 Output Description . . . 21

6 M_SIMAMP **22**
6.1 Input Cards . . . 22
6.2 Output Description . . . 23
6.3 Implementation . . . 24

7 M_TIMING **25**
7.1 Input Cards . . . 25
7.2 Output Description . . . 26
7.3 Implementation . . . 26

8 M_OVS **28**
8.1 Input Cards . . . 28
8.2 Output Description . . . 29
8.3 Algorithm . . . 29

9 S_READFILES **30**
9.1 Input Cards . . . 30

10 S_PORBITS **31**
10.1 Input Cards . . . 31

11 S_CROP **32**
11.1 Input Cards . . . 32

12 S_OVS **33**
12.1 Input Cards . . . 33

13 COARSEORB **34**
13.1 Input Cards . . . 34
13.2 Output Description . . . 34
13.3 Implementation . . . 35

14 COARSECORR **36**
14.1 Input Cards . . . 36
14.2 Output Description . . . 37
14.3 Implementation . . . 37
14.3.1 Method magspace . . . 37
14.3.2 Method magfft . . . 38

15 M_FILTAZI **39**
15.1 Input Cards . . . 39
15.2 Output Description . . . 40
15.3 Implementation . . . 40

16 S_FILTAZI **43**

17 FINE **44**
17.1 Input Cards . . . 44
17.2 Output Description . . . 46
17.3 Implementation . . . 47
17.3.1 magspace . . . 47
17.3.2 oversample . . . 47
17.3.3 magfft . . . 47

18 RELTIMING **48**
18.1 Input Cards . . . 48
18.2 Output Description . . . 49
18.3 Implementation . . . 49

19 DEMASSIST **50**
19.1 Input Cards . . . 50
19.2 Output Description . . . 51
19.3 Implementation . . . 52

20 COREGPM **53**
20.1 Input Cards . . . 57
20.2 Output Description . . . 58
20.3 Implementation . . . 59

21 RESAMPLE **61**
21.1 Input Cards . . . 61
21.2 Output Description . . . 62
21.3 Implementation . . . 63
21.3.1 Output formats . . . 64
21.3.2 Interpolation Kernels . . . 65

22 FILTRANGE **66**
22.1 Input Cards . . . 66
22.2 Output Description . . . 68
22.3 Implementation . . . 69
22.3.1 Method: porbits . . . 69
22.3.2 Method: adaptive . . . 69
22.3.3 Hamming filter . . . 71

23 INTERFERO **73**
23.0.4 NEW . . . 74
23.1 Input Cards . . . 74
23.2 Output Description . . . 75
23.3 Implementation . . . 76

24 COMPREFPHA **77**
24.1 Input Cards . . . 78
24.2 Output Description . . . 79

25 SUBTRREFPHA **80**
25.1 Input Cards . . . 80
25.2 Output Description . . . 81

26 COMPREFDEM **83**
26.1 Input Cards . . . 83
26.2 Output Description . . . 84
26.3 Implementation . . . 85

27 SUBTRREFDEM **87**
27.1 Input Cards . . . 87
27.2 Output Description . . . 87

28 COHERENCE **89**
28.1 Input Cards . . . 89
28.2 Output Description . . . 91
28.3 Implementation . . . 91

29 FILTPHASE **92**
29.1 Input Cards . . . 92
29.2 Output Description . . . 94
29.3 Implementation . . . 95
29.3.1 spatialconv . . . 95
29.3.2 spectral . . . 97
29.3.3 goldstein . . . 97

30 UNWRAP **99**
30.1 Input Cards . . . 99
30.2 Output Description . . . 100

31 DINSAR **102**
31.1 Input Cards . . . 102
31.2 Output Description . . . 103
31.3 Implementation . . . 103
31.3.1 Algorithm . . . 106

32 SLANT2H **108**
32.1 Input Cards . . . 108
32.2 Output Description . . . 110
32.3 Implementation . . . 110
32.3.1 Method ambiguity . . . 110

32.3.2 Method rodriguez . . . 111
32.3.3 Method schwabisch . . . 113
32.4 Comparison of the methods . . . 114

33 GEOCODE **117**
33.1 Input Cards . . . 117
33.2 Output Description . . . 118
33.3 Implementation . . . 119

A What's new? **122**
A.1 Version 4.02 . . . 122
A.2 Version 4.01 . . . 122

B Installation **124**
B.1 Installation of Doris . . . 124
B.1.1 Installation of the Doris core . . . 124
B.1.2 Installation of the SARtools . . . 125
B.1.3 Installation of the ENVISAT_tools . . . 125
B.1.4 Installation of the TERRASAR-X reader . . . 126
B.1.5 Starting Doris 126
B.1.6 Installation of utility scripts . . . 126
B.2 Additional programs . . . 127
B.3 Running the Doris software . . . 127
B.4 Viewing the results of Doris . . . 128
B.5 Trouble shooting . . . 129
B.5.1 General problems . . . 129
B.5.2 Matrix class troubles . . . 131
B.5.3 Some notes on installation on SGI . . . 131
B.5.4 Some notes on installation on Linux X86 . . . 133
B.5.5 Some notes on installation on Window running Cygwin . . . 133
B.6 List of files in archive . . . 133
B.7 List of routines + description . . . 134

C Utilities **138**
C.1 Packages . . . 138
C.2 Tools . . . 138
C.2.1 Installation of SARtools . . . 138
C.2.2 run script . . . 139
C.2.3 cpxfiddle . . . 139
C.2.4 cpx2ps . . . 140
C.2.5 phasefilt.doris . . . 141
C.2.6 flapjack . . . 141
C.2.7 cpxmult . . . 141
C.2.8 cpxdiv . . . 141
C.2.9 cpxconj . . . 142
C.2.10 floatmult . . . 142
C.2.11 wrap . . . 142
C.2.12 construct_dem.sh . . . 142
C.2.13 doris.process-reset.sh . . . 143
C.3 Completes for tcsh users . . . 143

D Definitions **144**
D.1 Constants . . . 144
D.2 Baseline . . . 144
D.3 Interferogram . . . 146
D.4 Polynomials . . . 148
D.4.1 Computation of coefficients . . . 148
D.4.2 Evaluation of polynomials . . . 149

D.5 (SAR) System parameters . . . 149
D.5.1 Azimuth . . . 149
D.5.2 Range . . . 150
D.6 Doppler, range and ellipsoid equations . . . 151
D.7 Orbit interpolation . . . 152
D.8 Format of the products . . . 153

E Matrix class **155**
E.1 Matrix class functions . . . 155

F Adding a module **158**
F.1 Formats . . . 158
F.2 Adding a Step . . . 159

Chapter 1

Introduction

This user's manual guides you through the radar interferometric processing with the Doris software. A user will be able to process ERS1/2, ENVISAT, JERS, RADARSAT, ALOS and TERRASAR-X data with the aid of this manual. The data must be in Single Look Complex format (SLC). Doris is not a SAR processor, i.e., you cannot process/focus RAW radar data. The manual also contains implementation specifications which can be helpful to a programmer for further development of Doris, or to understand WARNING and ERROR messages.

1.1 Overview of the InSAR Processing

A high-level description of the InSAR processing is shown in Figure 1.1 where a division in four blocks has been made. Block I depicts the preprocessing of the raw (radar and orbit) data to another format.

We won't be concerned with the raw orbit data, but it is included in the flow chart for completeness. The Delft precise orbits are used for ERS1/2 and Envisat, obtained via the getorb package (see, e.g., [Scharroo and Visser, 1998]). The second block consists of the co-registration where the slave image is aligned with the master image, and of the computation of the reference phase of the ellipsoid. In block III the interferometric products (complex phase image and coherence map) are computed. Finally in block IV the endproducts (e.g., a DEM or a deformation map) are computed.

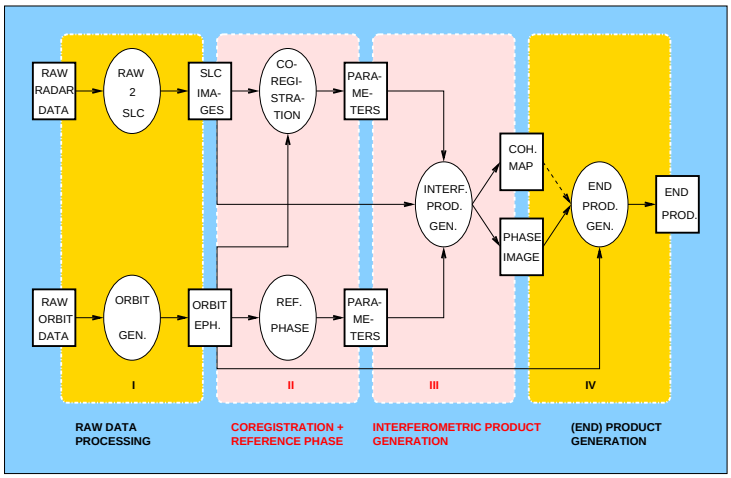

Figure 1.1: Coarse flow of the interferometric processing of SAR images.

The processing steps that are implemented in the Doris software are listed in the table below. Note that UNWRAP is NOT directly implemented in Doris.

To run each step, these names must be used as arguments for the PROCESS card, as explained in Chapter 2. A specific algorithm (module, method) can be selected with the cards that are special to this step, see the Chapters 3 to 33.

PROCESS (Chapter)	**Description**
M_READFILES (3)	Read the processing parameters from the SLC files for the master image.
M_PORBITS (4)	Retrieve the precise Delft orbital data records with the getorb package.
M_CROP (5)	Write the SLC data from paf format to disk in the 'raw' (pixel interleaved 2b/2b complex short integer) format.
M_SIMAMP (6)	Simulation of amplitude image based on DEM.
M_TIMING (7)	Estimation of timing error based on correlation between master amplitude and simulated amplitude.
M_OVS (8)	Oversampling of the master crop.
S_READFILES (9)	See M_READFILES.
S_PORBITS (10)	See M_PORBITS.
S_CROP (11)	See M_CROP.
S_OVS (12)	See M_OVS.
COARSEORB (13)	Compute the translation between master and slave with the orbits (precision 30 pixels).
COARSECORR (14)	Compute the translation between master and slave on pixel level by correlation technique.
M_FILTAZI (15)	Spectral filter for master image in azimuth (line) direction.
S_FILTAZI (16)	Spectral filter for slave image in azimuth (line) direction.
FINE (17)	Compute translation vectors over the total image on sub-pixel level.
RELTIMING (18)	Estimation of relative timing error between master and slave based on fine coregistration.
DEMASSIST (19)	DEM assisted coregistration.
COREGPM (20)	Compute the actual transformation (2d-polynomial) model for the alignment of the slave on the master image.
RESAMPLE (21)	Resample the slave image according to the transformation model of the coregistration.
FILTRANGE (22)	Spectral filter for master and slave image in range (pixel) direction.
INTERFERO (23)	Compute the (complex) interferogram.
COMPREFPHA (24)	Compute the reference phase of the ellipsoid to be subtracted from the interferogram (polynomial).
SUBTRREFPHA (25)	Subtract the reference phase of the ellipsoid from the interferogram.
COMPREFDEM (26)	Compute the reference phase of a DEM to be subtracted from the interferogram.
SUBTRREFDEM (27)	Subtract the reference phase of the DEM from the interferogram.
COHERENCE (28)	Compute the (complex) coherence map.
FILTPHASE (29)	Filter the the interferogram.
UNWRAP (30)	Unwrap the interferogram.
DINSAR (31)	3/4 pass differential interferometry.
SLANT2H (32)	Compute the heights of the pixels in the radar coded system.
GEOCODE (33)	Geocode the pixels (convert pixels from the radar coordinate system to a earth fixed reference system.)

1.1.1 Processing order

The processing order is not restricted, but, in general, the output of a step is the input for the next. The following flowchart (Figure 1.2) shows the general processing order. The black processing steps are obligatory to obtain a geocoded interferogram as end product. The dark gray steps are highly recommended, whereas the light gray steps are optional. Obviously, the modular structure of Doris enables the user to specify its own processing chain. The most important (intermediate) products are indicated on the right-hand side.

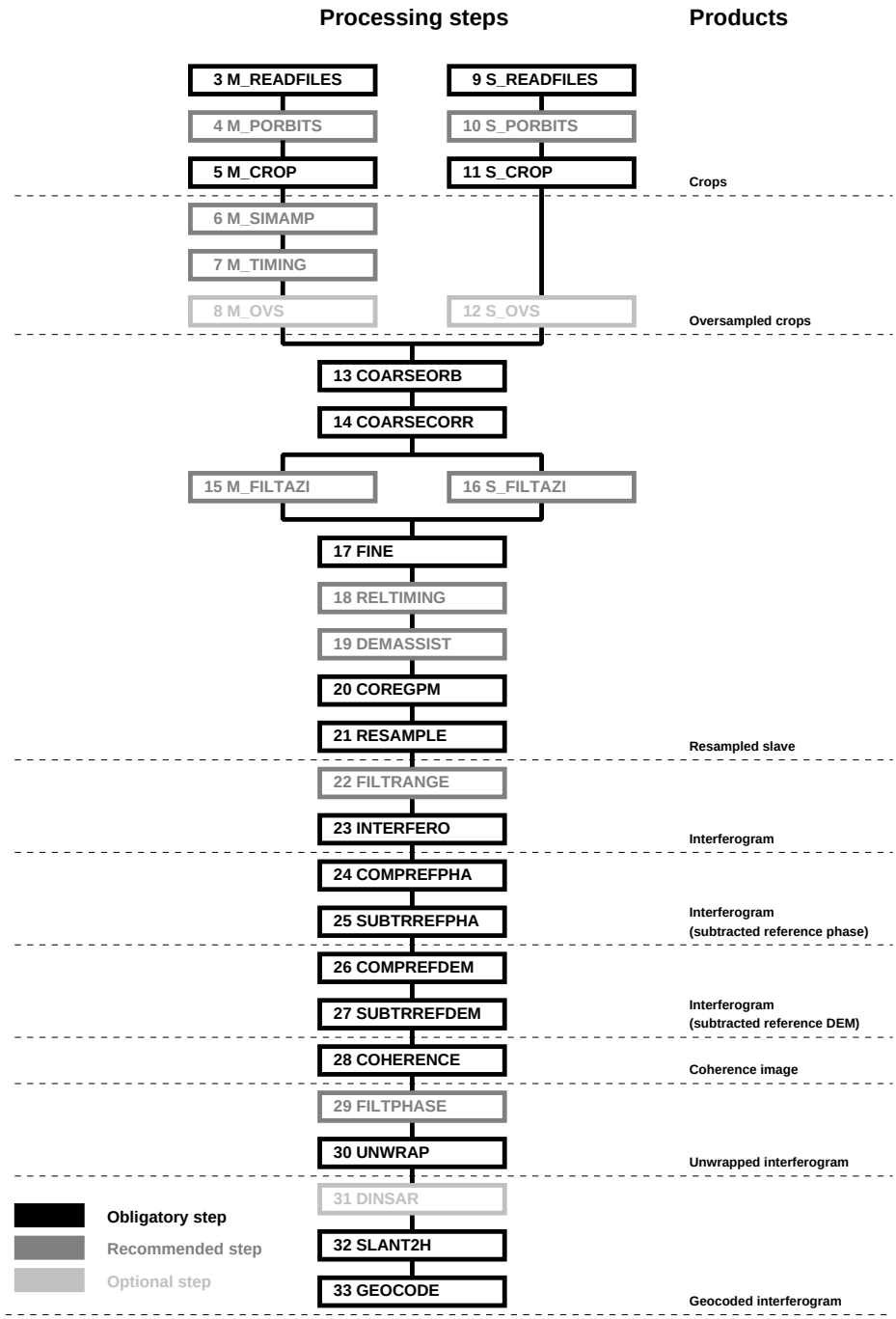

Figure 1.2: Processing flowchart for Doris. The black processing steps are obligatory to obtain a geocoded interferogram as end product. The dark gray steps are highly recommended, whereas the light gray steps are optional. Obviously, the modular structure of Doris enables the user to specify its own processing chain. The most important (intermediate) products are indicated on the right-hand side.

1.2 General considerations and conventions

In this section some important definitions are described that are used in the Doris software. This will clarify the terminlogy used. The general set up of the **input file** and **output files** is described as well.

After compilation with the Makefile, the executable is named: ”doris”. In this document therefor this name is used to refer to the executable. The command line options are:

- *doris -ver*
  return version number.
- *doris -h*
  return help (system call to shell script named helpdoris).
- *doris [file]*
  run, use input in ”file” (default: ”inputoptionsfile”.

It is advised also to compile an executable ”doris.debug” (with the Makefile). This version is somewhat slower and more verbose, but it can be used if something seems to go wrong with the normal executable, and it is not clear what. See also Annex B. We advice to use the utility scripts to generate **input files** and to run the processor.

Conventions are:

- We use the term *lines* to refer to the azimuth direction (slow time), and *pixels* for the range direction (fast time). (A *pixel* might also refer to an element, which will be clear from the context.) In the source code we frequently use the term azimuth *buffers* and range *blocks*.
- Our convention is to use **first lines**, **second pixels**, e.g., for the order of input arguments. The line direction (azimuth) corresponds to the vertical (y). The pixel direction (range) corresponds to the horizontal (x). Note that other software may use x before y.
- The first line (pixel) of an image is indexed as 1 (this may be a bit unusual). (In the software, the first index of an image (in a matrix) is equal to 0. To index a matrix, use MAT(y,x), i.e., as in linear algebra, Matlab, etc.)
- The name, format, and dimensions of the current master/slave/interferogram are stored in information structs that are filled by reading the corresponding **result files**. The files do not have a header.
- Generally all coordinates are in the master (radar) coordinate system. The first and last line are given, as well as multilook factors for both directions. If for example an interferogram is multilooked at the generation with a certain factor and later on again at the subtraction of the reference phase, then the first line is still the same, see Figure 1.3.
- In our view, the **output files** can only contain the results of one algorithm per step. So it is not possible to re-run a step (for example with another algorithm) without deleting the previous result. New users may be confused by this approach, but Doris should give an appropriate error message if it is attempted to run a step twice.
- Temporary files are created during the processing. Their names always start with ”scratch”. If such files are not removed by Doris, for example after an error in the processing, they can be savely removed by a *rm* command.

In the logfile additional information is written that is not sent to standard out or the **result files**, such as statistical information on a least squares estimate. (For example, we always inspect the correlation value for step COARSE_CORR, and the error between model and observations for reference phase computations in the log file.)

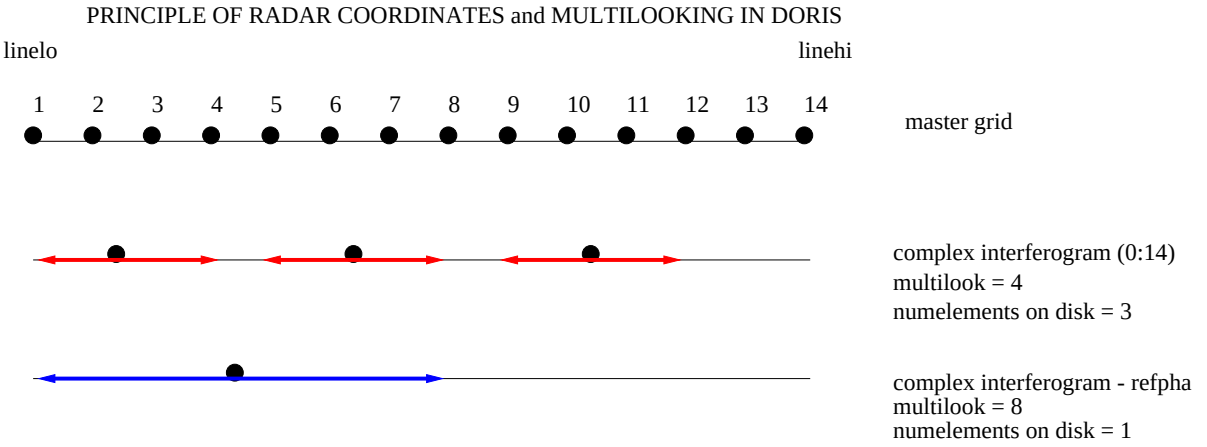

Figure 1.3: Principle of multilooking.

1.2.1 Inputfile

There is one ascii **input file** containing **cards** and zero or more **parameters** that controls the processor. A card is the first word on a line in the **input file**. The card and parameter(s) are delimited by blanks or tabs. There are mandatory cards (such as STOP at the end) and optional cards (because there are defaults for example for a filename for the output, or cards like COMMENT.) In this document the mandatory **CARDS** are in sans serif bold face, the optional CARDS are in normal style. Default parameters are underlined, and [optional] parameters are between square brackets.

The **input file** consists of a header and a tail. In the header, the general cards are placed (see Chapter 2), and in the tail the cards specific to a certain step are placed (described in the other chapters).

The order of the cards is not restricted (except the STOP card), though we advice to group them by processing step. Blank lines are allowed in the **input file**, but the line counter will not function properly in that case (which does not affect the processing in any way). We advice to place a comment on otherwise empty lines.

If (accidently) a certain card is used more than ones, then a WARNING is generated and the latter one is ignored (this behavior is not guaranteed, not true for PROCESS cards).

The case of cards and parameters is not restricted. We advice to use UPPER case (except for *comment* or *c*) for cards and lower case for parameters.

Text after the last expected parameter is ignored. Be careful with putting comments in like this if the number of parameters may be varied for a certain card. We normally do not make one big **input file**, but we use several small ones, for a group of processing steps. See also the run file in Annex B.

The examples in the next chapters will make this more clear. All keywords are described in this manual, and also in the interactive helpdoris script, and in the run script. It can occur that some keywords are not mentioned in the latter two. To obtain all possible cards, give the command:

```
grep keyword readinput.cc | grep else
```

1.2.2 Outputfiles

There are three ascii **output files**: one for (results of) processing steps specific to the master, one for the slave, and one for the rest of the processing (the 'products'). These files are referred to as master, slave and product **result file** (parameter files). For example, the wavelength of the sensor and the filename of the master image can be found in the **master result file** (and for the slave parameters in the **slave result file**), while coregistration parameters, which aren't unique to a particular image, can be found in the **products**

result file. These **output files** serve as input for Doris for running later steps. (Of course, a step also can generate binary data output. This is described in the following chapters.)

The **result files** all consist of a header and a tail, which grows with the processing.

In the header some general information and an overview of the processing is given with **process control flags**. (These flags do not imply a certain order.) By convention, each processing step can be run only once (0 or 1 in the **process control flag**), to avoid confusion on the correct/latest results are. (This implies that a result section in the tail has to be deleted, and the **process control flag** reset to 0, before running a step a second time.)

In the (growing) tail the results of the processing is stored. The **result files** are read again and the read parameters are used in the further processing. (In order to *trick* Doris to use other parameters then the ones that result from a previous processing step, simply edit the **result file**, e.g, in order to coregister complex interferograms.)

For the master **output file** the header with the information and the **process control flags** looks like:

```
====================================================
MASTER RESULTFILE:      master.res

Created by:
InSAR Processor:        Doris (Delft o-o Radar Interferometric Software)
Version:                Version 4.01 (19-DEC-2008) (optimal)
FFTW library:           used
VECLIB library:         not used
LAPACK library:         not used
Compiled at:            Dec 19 2008 17:26:52
By GNU gcc:             4.1.4
File creation at:       Fri Dec  19 19:08:21 2008

  ---------------------------------------------------
 | Delft Institute of Earth Observation and Space Systems |
 |        Delft University of Technology                |
 |       http://enterprise.lr.tudelft.nl/doris/         |
  ---------------------------------------------------

Start_process_control
readfiles:              0
precise_orbits:         0
crop:                   0
sim_amplitude:          0
master_timing:          0
oversample:             0
filt_azi:               0
filt_range:             0
NOT_USED:               0
End_process_control
```

The last flag (NOT_USED) is reserved for future use. In the **slave result file** the following line (and processing step) is **extra**:

```
resample:               0
```

and following lines are missing:

```
sim_amplitude:          0
master_timing:          0
```

The **products result file** is build up in the same manner. The **process control flags** in the header of the **products result file** are:

```
[SKIP] [SKIP]
```

```
Start_process_control
coarse_orbits:          0
coarse_correl:          0
fine_coreg:             0
timing_error:           0
dem_assist:             0
comp_coregpm:           0
interfero:              0
coherence:              0
comp_refphase:          0
subtr_refphase:         0
comp_refdem:            0
subtr_refdem:           0
filtphase:              0
unwrap:                 0
slant2h:                0
geocoding:              0
dinsar:                 0
NOT_USED2:              0
End_process_control

[SKIP] [SKIP]
```

The latter flag is reserved for future use. As already mentioned, after the **process control flags** the results of a (successfully ran) processing step are appended.

A section of the tail always starts with some lines like

```
*****************************************************************
*_Start_coarse_orbits:
*****************************************************************
```

After which the results for this processing step follow. A section always ends with a statement shown below. (This **End_step:_NORMAL** statement is important because the status of the process flag in the header is updated with it.)

```
*****************************************************************
* End_coarse_orbits:_NORMAL
*****************************************************************
```

Note that not all steps that are in the **process control flags** actually have to be implemented in Doris for the moment. (Unwrapping, extra flags)

Since only one result section is allowed for every processing step, it is not possible to re-run a certain step without editing the **result file**. The **process control flag** in the header has to be reset, and the total section in the tail (from Start_step to END:_NORMAL) has to be deleted, or commented out.

If the section is not deleted, Doris will likely exit, but if not, the further processing may be affected, because wrong values may be used (i.e., read from the **result file**).

It is of course possible to change the results (parameter values, for example correlation value for an estimated offset) in the **result files**, so that the altered value is used in the further processing. However, if you change the strings describing the output Doris will likely protest (i.e., hang or exit).

1.3 Outline of this document

In Chapter 2 the general purpose cards are described. All further chapters describe a certain processing step. The Chapter name is equal to the argument of the PROCESS card that should be given in the **input file** to switch on the processing of that step.

The step is introduced at the beginning of the chapter. In the first section the input cards are described and also an example **input file** is given. In the second section the output is described, as well with an example. In most chapters there is also a third section describing the implementation.

Cards that are mandatory are written in boldface sans serif. Optional cards are in normal sans serif font. Parameters are notated in italic, and defaults are underlined. Optional parameters are in between square brackets.

The definitions used in the software are described in Annex D. Here, amongst others, the baseline definition, the file formats used, and normalization of polynomials are described.

In Annex B the instalation of Doris is described. It also contains a small trouble shoot section.

Annex C shortly describe (third party) packages that should be installed for a complete version of the Doris software. Also some utilities we have developed are described.

The matrix class which comes with the Doris software is described in Annex E. This matrix class can be freely used in other non-commercial programs.

Finally, Annex F describes how to add a module to the Doris software. Extention of the software is encouraged.

Chapter 2

General Cards

This chapter deals with input cards that are not specific for a certain processing step, the *general* input cards. For example, such a card could specify whether you like to do batch processing or interactive processing. These cards are best placed at the start of the **input file**.

They do not generate any specific output. An example of the (header of the) **input file** is given in section 2.2.

2.1 General Input Cards

\\ [...]
After this card everything up to a newline is ignored. A space after \\ is not required.

[...]
After this card everything up to a newline is ignored. A space after # is not required.

C [...]
After this card everything up to a newline is ignored. A space after this card is not required.

COMMENT [...]
After this card everything up to a newline is ignored. A space after this card is not required.

SCREEN *DEBUG* | *INFO* | *PROGRESS* | *WARNING* | *ERROR*
This card controls the level of standard output. It is recommended to start with this card, since it is in effect only after it is read.

BEEP *OFF* | *WARNING* | *ERROR* | *PROGRESS* | *ON*
This card controls the level of beeping.

BATCH [*ON* | *OFF*]
Specifies to run the processor in non-interactive mode. If this card is omitted then the processing is done in interactive mode, asking to press a key before each step. BATCH can be specified without arguments, which means non-interactive processing. (If there is an ONLYPROCESS card present, this forces BATCH ON).

OVERWRITE [*ON* | *OFF*]

Specifies whether or not to overwrite existing files. If this card is omitted files are not overwritten; if no parameter is given it defaults to ON (do overwrite).

PREVIEW [*OFF* | *ON* | *XV*]

Specifies whether or not to generate SUNraster preview files with the help of the utility program cpxfiddle (download and install seperately from Doris website.). Default is OFF since this program may not be installed. If ON, shell scripts are created in the working directroy which create the SUNraster file if run. If XV is given, also the command is given to view the generated file with xv.

LISTINPUT [*ON* | *OFF*]

Specifies if the **input file** has to be copied to the logfile. If this card is omitted then input is not copied. if no parameter is given it defaults to ON (do copy).

MEMORY *500*

With this card the user can indicate the maximum amount of memory to be used by the processor (in Megabytes). It is advised setting this lower than the maximum available amount, because it may be somewhat inaccurate (up to a factor 2, particularly due to temporary copies created by the copy constructor). A lot of routines actually try to use a minimum of memory, even if this card is set to a large value.

PROCESS *M_READFILES* | *M_PORBITS* | *M_CROP* | *M_SIMAMP* | *M_TIMING* | *M_OVS* | *M_FILTAZI* | *FILTRANGE* | *S_READFILES* | *S_PORBITS* | *S_CROP* | *S_OVS* | *S_FILTAZI* | *COARSEORB* | *COARSECORR* | *FINE* | *RELTIMING* | *DEMASSIST* | *COREGPM* | *RESAMPLE* | *INTERFERO* | *COMPREFPHA* | *SUBTRREFPHA* | *COMPREFDEM* | *SUBTRREFDEM* | *COHERENCE* | *FILTPHASE* | *DINSAR* | *UNWRAP* | *SLANT2H* | *GEOCODE*

With this card the processing steps that have to be processed can be switched on. More than one PROCESS card can be specified in the **input file**. An ONLYPROCESS card overrides possible PROCESS cards. Description of these steps can be found in the introduction, chapter 1, and in the following chapters. At least one PROCESS or ONLYPROCESS card is mandatory.

ONLYPROCESS *same arguments as* **PROCESS** *card*

With this card a processing step that has to be processed can be switched on. Overrides possible PROCESS cards. This card also automatically switches: BATCH ON. At least one PROCESS or an ONLYPROCESS card is mandatory.

LOGFILE *log.out*

Output filename for the logfile.

M_RESFILE *master_result.out*

Output filename for the **master result file**.

S_RESFILE *slave_result.out*

Output filename for the **slave result file**.

I_RESFILE *interferogram.out*

Output filename for the **products result file**.

ORB_INTERP *POLYFIT [*[*DEGREE*]*]* | [*SPLINE*]

Orbit interpolation method. Defaults to a polynomial of degree numberofdatapoints-1, but smaller than degree 5 (order 5). Optionally, the DEGREE can be given (¡=numberofdatapoints-1). The x,y,z are independently interpolated, the velocities are estimated from the position. If method SPLINE is selected, natural cubic splines are used. This may be inaccurate if there are only a few orbit datapoints. (default interpolation, not approximation for polyfit is used to go smoothly through the datapoints since the points do probably not contain noise since they are already the result of an orbit propagator somewhere. It is not adviced really to use a DEGREE smaller than the maximum possible, except if it gets too large to avoid oscillations.)

DUMPBASELINE *0 0*

Dump the baseline parameters for a grid of 0 lines by 0 pixels as INFO to stdout. The baseline is only evaluated after the orbits are known. The perpendicular baseline to the reference ellipsoid is also computed as a 2D polynomial of degree 1. And also theta as function of azimuth line and range (though it hardly varies over azimuth).

HEIGHT *0.0*

Average terrain height above WGS84. This can be used in future to correct the unwrapping for the integration constant. Now it is only used if GEO card is used for cropping.

TIEPOINT *lat lon hei*

Coordinates of a point in lat lon hei in WGS84. For now, only informational. The point is converted to pixel/line coordinates, and the interferometric phase is computed, etc.

M_RG_T_ERROR [*0.0*]

Range timing error for master. One-way in seconds. Use this card for example to calibrate the geo-referencing using a corner reflector with known coordinates. Can also be used to “shift” the DEM with respect to the interferogram in step COMPREFDEM. A shift of one (non-mulitlooked or oversampled) pixel corresponds to a one-way timing error of 1/(2*RSR). For ers this is approximately M_RG_T_ERROR=0.00000002637 seconds. By multiplication of the signal velocity speed of light (3e8) this amount in seconds can be converted to the slant-range resolution (i.e., pixel posting) of 7.9 meter.

M_AZ_T_ERROR [*0.0*]

Azimuth timing error for master. Use this card to account for timing errors in azimuth direction. Card can be used to shift a DEM in azimuth direction. Note that such a shift may indicate incorrectly estimated Doppler.

S_RG_T_ERROR [*0.0*]

Range timing error for slave. See M_RG_T_ERROR. Since the geometry of the interferogram is related to the master this card has not a large effect.

S_AZ_T_ERROR [*0.0*]

Azimuth timing error for slave. See M_AZ_T_ERROR. Since the geometry of the interferogram is related to the master this card has not a large effect.

STOP

After this card the **input file** is no longer interpreted. This card is mandatory.

2.2 Example General Input Cards

```
c ************************************************************
```

```
c *  Doris \inputfile generated by: run at: Nov 27, 2000 (Monday) *
c ***************************************************************
c *
c *  Filename:        Inputfiles/input.s_initial
c *  Author:          Doris User
c *  Master:          23185
c *  Slave :          03512
c *  Baseline:        170 m
c *  Remarks:         Test: s2h routine (exact)
c *
c ***************************************************************
c
 c
 comment  ___general options___
 c
SCREEN          debug                           // level of output to standard out
MEMORY          150                             // MB
OVERWRITE                                       // overwrite existing files
BATCH                                           // non-interactive
 c LISTINPUT OFF                                // prevents copy of this file to log
 c
PROCESS          m_readfiles                    // read parameters
PROCESS          m_porbits                      // obtain precise orbits
PROCESS          m_crop                         // crop data to internal format
 c                                              //
 c                                              //
 comment  ___the general io files___            //
 c                                              //
LOGFILE         log.out                         // log file
M_RESFILE       master.out                      // parameter file
S_RESFILE       slave.out                       // parameter file
I_RESFILE       interferogram.out               // parameter file
 c                                              //

[SKIP][SKIP]
...
... more cards specific to step specified by (ONLY)PROCESS cards,
... see next chapters for details on these cards.
...
[SKIP][SKIP]

STOP
```

Note that ”//” is not a delimiter for comments, text after the last expected parameter is simply ignored.

Chapter 3

M_READFILES

In this chapter the processing of step M_READFILES is described. It can be selected by a PROCESS M_READFILES line in the **input file**. This is the first step if the ERS1/2 SLC images are processed.

The SLC leader, volume and (header of the) data file are read, and relevant parameters are written to the **master result file** specified by the general card M_RESFILE. These parameters are used in the further processing. Currently, ERS1/2 SLC and ENVISAT SLC files can be read. If the output of this step is mimicked, Doris can be tricked to process the other steps. The sole purpose of this step is to create **result file** where relevant parameters are stored (PRF, wavelength, etc.), also see the example in the next section.

3.1 Input Cards

M_IN_METHOD *ERS* | *ASAR (ENVISAT)* | *RSAT (RADARSAT)* | *ATLANTIS* | *JERS* | *ALOS* | *TSX (TERRASAR-X)*

Method selector to read ERS, ENVISAT, RADARSAT, JERS, ALOS or TERRASAR-X header. Note that both master and slave need to be acquired by the same sensor in principle. JERS simply uses ERS programs, ATLANTIS (sar processor) uses the ceos reader for RSAT, and will write this in the Product Type Specifier field. RSAT must be tested, problems may be orbit data. In later steps, the Product field is read, and the CROP step uses the appropriate function automatically (Envisat, ERS/JERS, RSAT/ATLANTIS).

M_IN_DAT *filename*

The filename of the SLC data file. This is the only file required for method ASAR (ENVISAT).

M_IN_LEA *filename*

The filename of the SLC leader file. Not used for method ASAR (ENVISAT).

M_IN_VOL *filename*

The filename of the SLC volume file. Not used for method ASAR (ENVISAT) and TSX (TERRASAR-X).

M_IN_NULL *filename*

The filename of the SLC null file. This may be a dummy name since it is not used.

An example of the input cards for this step is given below. This example can be inserted in the general cards described in Section 2.2.

```
 c
 c
 comment ___READFILES___
 c
M_IN_METHOD    ERS
M_IN_VOL       /cdrom/scene1/vdf_dat.001       // name of volumefile
M_IN_LEA       /cdrom/scene1/lea_01.001        // name of leaderfile
M_IN_NULL      dummy                           // name of nullfile
M_IN_DAT       /cdrom/scene1/dat_01.001        // name of datafile
```

3.2 Output Description

The **process control flag** at the start of the **master result file** is switched to 1 at successful exit.

```
readfiles:              1
```

Example of output of this step (in **master result file**). (The positioning data of the platform from the leader file has been deleted in this example. This happens automatically after step M_PORBITS (getting the precise orbits)). This output is appended to the **master result file**.

```
*****************************************************************
*_Start_readfiles:
*****************************************************************
Volume file:                                /cdrom/scene1/vdf_dat.001
Volume_ID:                                   1
Volume_identifier:                           0004093800014027
Volume_set_identifier:                       19950830 9491991
(Check)Number of records in ref. file:       26558
Product type specifier:                      PRODUCT:ERS-1.SAR.SLC
Location and date/time of product creation: IPAF 24-07-1998
Scene identification:                       ORBIT 21567 DATE 30-08-95
Scene location:                        FRAME 2781 LAT:40.94 LON:14.03
Leader file:                                 /cdrom/scene1/lea_01.001
Scene_centre_latitude:                             40.9380000
Scene_centre_longitude:                            14.0270000
Radar_wavelength (m):                               0.0566660
First_pixel_azimuth_time (UTC):              30-AUG-1995 09:49:20.453
Pulse_Repetition_Frequency (actual, Hz):        1679.9020000
Total_azimuth_band_width (Hz):                  1378.0000000
Weighting_azimuth:                           HAMMING
Xtrack_f_DC_constant (Hz, early edge):           437.9780000
Xtrack_f_DC_linear (Hz/s, early edge):          7154.0000000
Xtrack_f_DC_quadratic (Hz/s/s, early edge):-380000000.00000
Range_time_to_first_pixel (2way) (ms):             5.5458330
Range_sampling_rate (leaderfile, MHz):            18.9624680
Total_range_band_width (MHz):                     15.5500000
Weighting_range:                                  HAMMING
Datafile:                                    /cdrom/scene1/dat_01.001
Number_of_lines_original:                         26183
Number_of_pixels_original:                        4900
*****************************************************************
* End_readfiles:_NORMAL
*****************************************************************
```

Note that Product specifies ”ASAR” for ASAR, which is used later.

A number of these lines is only for your information. The lines after *Leader file:* are used for the further processing (except Weighting identifiers). These strings may not be altered. We encountered a problem once when there was a blank in the UTC time, instead of a zero, but this should be fixed.

The logfile shows more details. Also some information is echoed to the screen (as "INFO: .."), such as the Doppler centroid frequency, evaluated at some ranges, the corners of the images in latitude longitude, etc.

Defaults for parameters (slcimage.cc; Doris can still crash if not correct, e.g., the approximate coordinates of the scene):

```
wavelength = 0.0566660.;          // [m] default ERS2
t_range1   = 5.5458330/2.0e3;     // [s] one way, default ERS2
prf        = 1679.902.;           // [Hz] default ERS2
abw        = 1378.0;              // [Hz] default ERS2
rsr2x      = 18.9624680*2.0e6;    // [Hz] default ERS2
rbw        = 15.55e6;             // [Hz] default ERS2
```

3.3 Implementation

Three basic data formats are supported: CEOS (ERS/JERS/ALOS/RADARSAT), N1 (ENVISAT), and COSAR/XML (TERRASAR-X). By specifying the M_IN_METHOD the correct reader is selected.

3.3.1 Changes for X86 platforms

Version 2_4 onward can be (easily) compiled on Linux systems. For little endian machines like (intel) PC's this means the byte order is different. Since the record length in the leader, volume, and data file are stored as B4 (4bytes unsigned integers), we had to use the function ntohl, see the manual pages. routines readvolume, readleader, readdat.

Also the SLC data itself in the datafile is stored as 2x 2B short signed integer byte data. (real part, imaginary part, real, imaginary, real, imag, ...). We transform the data that is read from the data file with the function htons if __X86PROCESSOR__ is defined (see the Makefile or source code) for more information.

Chapter 4

M_PORBITS

In this chapter the step M_PORBITS is described. This step should be run after READFILES, because in that step the azimuth time is written to the **master result file** from from the SLC leader file.

We use the DEOS fortran program **getorb** for obtaining the precise orbits. (This program has to be installed separately, see `http://www.deos.tudelft.nl/ers/precorbs/` or [Scharroo and Visser, 1998].) This step actually is only a system call to getorb, and converts the output to a 4 column table: secofday,x,y,z.

It requires the Orbital Data Records (ODR files) to be in an archive directory. The *arclist* file (which should be downloaded together with the ODR archives) has to be present in this directory. The ODR files have to be untarred and unzipped as from version 2.5.

This step introduces a section in the **result file** where the ephemerides are placed, and it deletes the ephemerides from the SLC leader file, obtained by the processing step M_READFILES (if there was such a section). The ephemerides (x,y,z) span the time 4 seconds before the first line and 4 seconds after the last line by default. The time is, and should be, in *seconds of day*. The time interval is 1 second by default. Natural cubic splines are used for interpolation, and the boundary conditions may affect the interpolation if only a few datapoints are used, e.g., 5 points with a time interval of 30 seconds. We nowadays use a time interval of 30 second, and approximtely 21 points. This implies that only spline (degree 3 piecewice polynomial) is used for the whole image, which gives better results for, e.g, reference phase computation. The interpolation errors in Doris are probably due to interpolation of interpolated values of getorb, which output format is in 3 digits.

If you want to use other ephemerides you can simply insert them in the **result file** in the format described in section 4.2. You will have to correct the number of POINTS in the **result file**. Note that the orbit system is WGS84 (only).

4.1 Input Cards

M_ORBDIR *directory name*
the tar archive directory name for the Delft Orbital Data Records.

M_ORB_INTERVAL *1*
Time in seconds between ephemerides.

M_ORB_EXTRATIME *3*

Time in seconds before first and after last line to output ephemerides. Since interpolation is done with natural cubic splines, it is advised to have at least 3 extra data points before the first and after the last line. To use a single polynomial of degree 3 for interpolation of the orbit for the full scene, select a time interval of 20 seconds, and, for example, extra time of 200 seconds.

M_ORB_DUMP *delta t*

Write interpolated orbit to ascii **output file** "masterorbit.dat". With delta t seconds interval between ephemerides. Time interval between t_0 and t_N of the precise ephemerides. output is: *t,x,y,z,xdot,ydot,zdot,xddot,yddot,zddot*. If compiled with __DEBUG defined, then also the matrices for spline interpolation are dumped.

Example of the cards for this step:

```
 c
 c
 comment   ___PORBITS___
 c
M_ORBDIR          /data/delftorbits/ERS1/
M_ORB_INTERVAL    1
M_ORB_EXTRATIME   6
c M_ORB_DUMP      0.1
```

4.2 Output Description

If a normal termination of this step, then the process flag at the start of the **result file** is switched to 1:

```
precise_orbits:         1
```

The output of this step is written in the section: precise_datapoints. This section looks like the following. It is important that all lines are present following NUMBER_OF_DATAPOINTS: 23.

```
*****************************************************************
*_Start_precise_orbits:
*****************************************************************
 t(s)    X(m)    Y(m)    Z(m)
NUMBER_OF_DATAPOINTS:                  23
35360.000000    5161849.442     1645908.227     4678710.927
35361.000000    5166975.704     1645589.230     4673176.199
35362.000000    5172096.340     1645267.708     4667636.387
35363.000000    5177211.345     1644943.664     4662091.497
35364.000000    5182320.713     1644617.098     4656541.536
35365.000000    5187424.437     1644288.011     4650986.509
35366.000000    5192522.513     1643956.405     4645426.423
35367.000000    5197614.933     1643622.281     4639861.283
35368.000000    5202701.692     1643285.640     4634291.095
35369.000000    5207782.784     1642946.483     4628715.867
35370.000000    5212858.204     1642604.811     4623135.603
35371.000000    5217927.945     1642260.626     4617550.309
35372.000000    5222992.002     1641913.928     4611959.992
35373.000000    5228050.368     1641564.720     4606364.658
35374.000000    5233103.038     1641213.002     4600764.313
35375.000000    5238150.007     1640858.775     4595158.964
35376.000000    5243191.267     1640502.040     4589548.615
35377.000000    5248226.814     1640142.800     4583933.273
35378.000000    5253256.642     1639781.054     4578312.945
```

```
35379.000000    5258280.744     1639416.805     4572687.636
35380.000000    5263299.115     1639050.053     4567057.352
35381.000000    5268311.750     1638680.800     4561422.100
35382.000000    5273318.642     1638309.047     4555781.886

*****************************************************************
* End_precise_orbits:_NORMAL
*****************************************************************
```

The time is in seconds of day. which can be computed as: *fractional_day * 60*60*24* or *hours*60*60 + min*60 + sec*.

If card M_ORB_DUMP then an ascii file ”masterorbit.dat” is written with the computed *t, x, y, z, xdot, ydot, zdot, xddot, yddot, zddot*.

4.3 Implementation

Based on the UTC time of the image and the PRF and number of lines, basically the program getorb is called through a UNIX system call. Functions from the standard library *ctime* (or time.h) are used for the time conversions. This call is echoed to the screen as DEBUG. The commands can be executed stand-alone as well.

```
%getodr 950727094923,950727094945,1 /data/orbits/ERS2.ARCS > dummyout
%
%(...read in ODR file name from dummyout (ODR.422))
%untar /data/orbits/ERS1.ARCS ODR.422
 getorb 950727094923,950727094945,1 /data/delftorbits/ERS1/ \
    > scratchorbit
```

The ephemerides are first written to a dummy file (named ”scratchorbit”) and later placed in the **result file** (without the velocity output, only t,x,y,z). It has been noted that sometimes this scratch file is not automatically removed. This file can be savely removed by hand.

Natural cubic splines are used for the interpolation so it may be wise to have a short time-interval and some data before the first and after the last line.

If there is a section with the ephemerides of the SLC leaderfile in the **master result file**, then this section is removed.

In the routine splineinterpol, file utilities.c, where the coefficients are computed, _NATURALSPLINE_ is defined. This sets the boundary condition to use zero second derivative at the borders. Otherwise, the first derivative is set to a specified value. This does not seem to make a big difference.

Chapter 5

M_CROP

In this chapter the processing of step M_CROP is described. This step normally is the third one that is run, after M_READFILES and M_ORBITS. It requires the SLC data file on disk or cdrom. For ENVISAT, a utility is called that does the work.

In this step the SLC datafile is put on disk in a raw (pixel interleaved, 2x2byte signed (short) integer complex) format. (The reason for this step is that we normally work with the SLC images on cdrom, and that we want to have the files on disk to perform operation requiring both the images. It also serves as a common format for different input.)

A few checks are performed regarding the number of lines, which is written in the header of the SLC data file as well as in the leader file. The image is read/written line by line, no data conversion takes place (though a cutout can be made). If you are working on a little endian platform (X86 PC) then the data is converted from big endian (which is the CEOS format).

5.1 Input Cards

M_CROP_IN *filename*
Filename of the SLC data file.

M_CROP_OUT *master.raw*
Filename of the raw data **output file**.

M_DBOW *linelow linehi pixellow pixelhi*
Master output window. You can make a cutout of the image with this card. Hi values larger than the size of the image are reset to the maximum. If card omitted it defaults to total image. line/pixel 1 refers to the the first line/pixel.

M_DBOW_GEO *lat_0 lon_0 height width*
Master output window. Alternative to and overrides normal DBOW card. You can make a cutout of the image with this card. latitude of the center pixel of the desired crop, longitude (in decimal degrees, WGS84 system of orbit), then height, width in pixels. For approximately square areas, heights should be a factor 5 of width for ERS.

Example input cards for this step:

```
c
c
comment ___CROP___
```

```
 c
M_CROP_IN    /cdrom/SCENE1/DAT_01.001
M_CROP_OUT   Output/21066.raw
c M_DBOW       1 5000 1 1000              // linelow hi pixellow/hi
M_DBOW_GEO   52.6734 5.4342 10000 2000  // lat_0[deg], lon_0, height, width[pix]
```

5.2 Output Description

The **process control flag** at the start of the **result file** is switched to 1 at successful exit.

```
crop:                   1
```

The output section in the **result file** will resemble the following.

```
*****************************************************************
*_Start_crop:                        master
*****************************************************************
Data_output_file:                              Output/21066.raw
Data_output_format:                            complex_short
First_line (w.r.t. original_master):           1
Last_line (w.r.t. original_master):            5000
First_pixel (w.r.t. original_master):          1
Last_pixel (w.r.t. original_master):           1000
*****************************************************************
* End_crop:_NORMAL
*****************************************************************
```

If the SLC data is already on disk, for example because the SAR processing was done, this section will have to be simulated. (As well as the result from READFILES.) The format complex_real4 is available.

Note that the byte order must be the same as the (host) platform order. This means that if data is copied from big endian platforms, they have to be swapped. Use for example a dd command like:

```
dd if=/cdrom/file.slc of=file.slc conv=swab
```

or use gmt (-Zh for short, -Zf for float, see man pages):

```
xyz2grd /cdrom/file.slc -Zh -Sfile.slc
```

Chapter 6

M_SIMAMP

In this chapter the processing of step **M_SIMAMP** is described, where the amlitude of the master image is simulated. This step requires an external digital elevation model (DEM) such as SRTM (Shuttle Radar Topography Mission) elevation data. This step can be performed **optionally** to simulate the master amplitude [Eineder, 2003] and should be done after the M_CROP step. In the following step M_TIMING, the synthetic amplitude can be used to compute the absolute timing error of the master acquisition.

We compute the synthetic amplitude for a given master acquisition using orbital information and topographic data. SRTM can be used to obtain the topography in 3 arcseconds (Near global: $\sim$ 90 m. at the equator) or 1 arcsecond (USA only) resolution. The input DEM file must have the byte order of your platform in order to extract correct elevation value, see SAM_IN_FORMAT option for details. The DEM should be in the WGS84 system (same as the orbit ephemerides) The Doris distribution contains the utility *construct_dem.sh* to download and prepare SRTM data (see Section C.2.12).

6.1 Input Cards

SAM_IN_DEM *filename*
filename of input DEM (gtopo30). File is assumed to be stored in a raster. Major row order. from North to South, line-by-line. See also internet links at Doris home page for available DEMs.

SAM_IN_FORMAT *I2* | *I2_BIGENDIAN* | *R4* | *R8*
format of input DEM on file (signed short for gtopo30, or real4, or real8; input matrix is raw binary data w/o header, endianness of host platform is assumed, except for I2_BIGENDIAN).

SAM_IN_SIZE *6000 4800*
Number of rows and columns of input DEM file. Default is set to tile w020n90.DEM.

SAM_IN_DELTA *0.0083333333333333333* [*deltalon*]
Grid spacing of input DEM in decimal degrees, latitude longitude. Default is equal gridspacing, default set to tile w020n90.DEM.

SAM_IN_UL *89.995833333333333 -19.995833333333333333333*
Coordinates of UL (upperleft) corner in decimal degrees, latitude [-90, 90] longitude [-180, 180]. Default is set to tile w020n90.DEM. It is interpreted as max-latitude, min-longitude in source.

SAM_IN_NODATA *-9999*

Identifier to ignore data in input DEM with this value. Default is set to tile w020n90.DEM.

SAM_OUT_FILE *master.sam*
Filename of the output simulated amplitude.

SAM_OUT_DEM *filename*
Request optional debug output to float file of input DEM per buffer, cut to the interferogram window. Info on these files is written as DEBUG. Default is set to dem-crop_sam.raw

Example input section:

```
c       ___      ___
comment ___SIMAMP___
c
SAM_IN_DEM        final_WAna.dem           // input DEM
SAM_IN_FORMAT     r4                       // real4 format
SAM_IN_SIZE       3601 3601                // extend of the DEM(l,p)
SAM_IN_DELTA      0.000833333 0.000833333  // 3 arcseconds
SAM_IN_UL         40  27                   // the center coordinates
                                           // of the UL corner pixel
SAM_IN_NODATA     -32768                   // ignored value
SAM_OUT_FILE      Outdata/18226.sam        // synthetic amplitude
SAM_OUT_DEM       Outdata/dem_sam.raw      // cropped DEM for
```

6.2 Output Description

At successful exit the **process control flag** is switched on:

```
sim_amplitude:            1
```

The output (in the **master result file**) looks like:

```
*_Start_sim_amplitude:
************************************************************
DEM source file:                        final_WAna.dem
Min. of input DEM:                      11
Max. of input DEM:                      2521
Data_output_file:                       Outdata/18226.sam
Data_output_format:                     real4
First_line (w.r.t. original_master):    1
Last_line (w.r.t. original_master):     29650
First_pixel (w.r.t. original_master):   1
Last_pixel (w.r.t. original_master):    4992
Multilookfactor_azimuth_direction:      1
Multilookfactor_range_direction:        1
Number of lines (multilooked):          29650
Number of pixels (multilooked):         4992
************************************************************
* End_sim_amplitude:_NORMAL
************************************************************
```

The output in the logfile is more verbose, specifying the results of the intermediate steps. Also go over the standard out in case of problems, with the SCREEN set to DEBUG level.

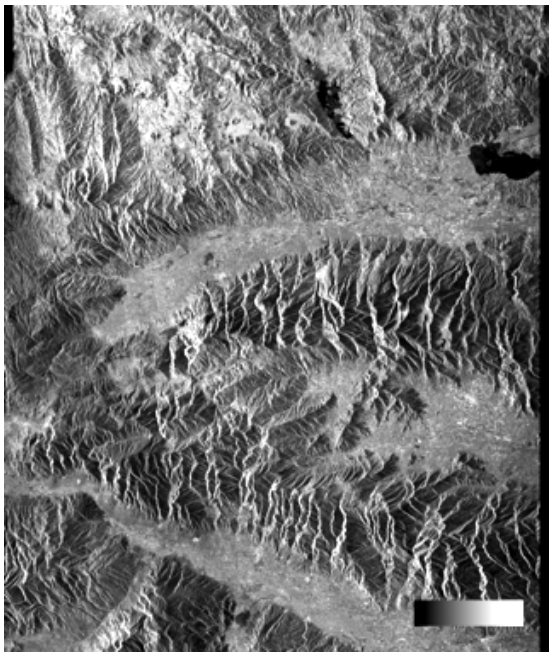

Figure 6.1: The orginal multilooked amplitude of the master image.

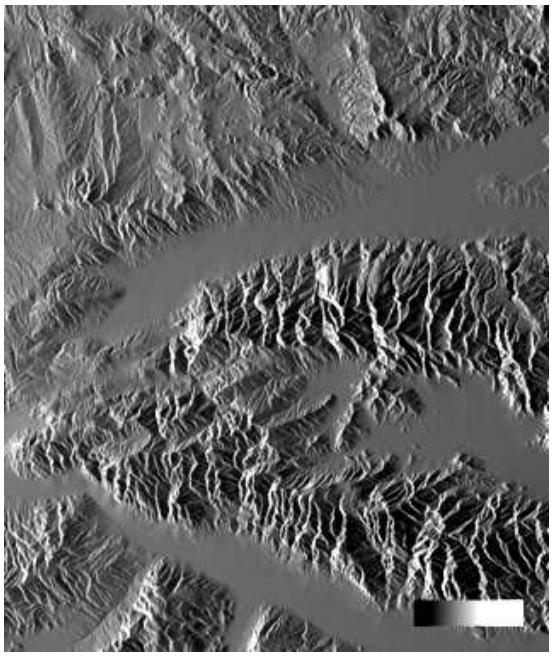

Figure 6.2: The simulated amplitude of the master image.

6.3 Implementation

The simulation of master amplitude is computed in three steps:

First, the DEM is radarcoded to the coordinate systems of the master acquisition. For each DEM point, the master coordinates and the look angle (θ) are computed and saved to temporary files. Both the master coordinates and look angle are real valued.

Second, the look angle and topographic heights are interpolated to the integer grid of master coordinates. A linear interpolation based on a Delaunay triangulation is used. The software package *Triangle* for the Delaunay triangulation is kindly made available by Jonathan Shewchuk [Shewchuk, 1996, Shewchuk, 2002], see also `http://www.cs.cmu.edu/~quake/triangle.html`. The interpolated topographic height and look angle are then used to compute the local incidence angle (θ_{loc}) for each point. Using the local incidence angle we obtain the synthetic amplitude by

$$\text{synthetic\_amplitude} = \sin(-\theta_{loc}) + 1. \tag{6.1}$$

The $+1$ is applied to obtain positive numbers.

Finally, the obtained simulated amplitude per master pixel are saved to a file and used in the **M_TIMING** step to obtain absolute timing error of the master acquisition.

Chapter 7

M_TIMING

In this chapter the processing of step **M_TIMING** is described. The absolute timing error between DEM and the master is computed using the simulated amplitude and the result of the coarse coregistration. Therefore, this step can only be run after the steps **M_CROP** and **M_SIMAMP**. The timing error in azimuth as well as in range direction is estimated.

During the coregistration, the master is aligned with respect to the DEM based on the simulated amplitude image, producing a single offset for the whole image. Because the resolution of a DEM is typically coarser (e.g., SRTM3: 90 m) than that of a radar image and the radar (ground) resolution differs between azimuth and range direction (e.g., $\sim$4 m in azimuth and $\sim$20 m in range for ERS1/2 and Envisat), the sensitivity in coregistration differs between the two directions (measured in resolution cells). As a result, a non-square correlation window may be more suitable. For example, for ERS1/2 we apply a 256 lines by 128 pixels correlation window.

This step updates the master acquisition azimuth and range times, respectively. Therefore, it affects steps whenever the master timing is used such as the coregistration using a DEM (**DEMASSIST** step) and the computation of reference phases (**COMPREFPHA** and **COMPREFDEM** steps).

7.1 Input Cards

MTE_METHOD *magfft* | *magspace*
Method selector for this step. Either perform the correlation computation on the magnitude images in the space or in the spectral domain.

MTE_IN_POS *filename*
Input filename for ASCII file with positions in original master system to place windows for correlation computations.

MTE_NWIN *16*
Number of windows to be distributed over the total image to estimate the offset. Should be at least 5 or so because the most consistent estimate is selected. This card is ignored if MTE_IN_POS is set. Only 1 large window could be used, e.g., of size 1024x1024.

MTE_WINSIZE *256 128*
Size of the window in lines pixels. For method in space domain it defaults to 256 128 For method in space domain it is converted to odd numbers if necessary.

MTE_ACC *32 32*

ONLY for method in space domain. Accuracy to search within for maximum correlation. For fft method it automatically equals half of the **MTE_WINSIZE**. In case of magspace, the DEM window size is extended by 2xMTE_ACC.

MTE_INITOFF *0 0*
Initial offset for coarse coregistration. The given offsets will shift master over simulated amplitude. Use for debuging: 0 0 by default

Example input cards for this step:

```
c       ___                              ___
comment ___COMPUTE MASTER TIMING ERROR___
c
c MTE_METHOD       magfft       // computes faster than magspace
MTE_METHOD        magspace     // default.
MTE_ACC           128 32       // only for magspace
MTE_NWIN          256          // number of large windows
MTE_WINSIZE       256 128      // rectangular window
MTE_INITOFF       0 0          // initial offset
```

7.2 Output Description

The **process control flag** at the start of the **master result file** is switched to 1 at successful exit.

```
m_timing:              1
```

Example of output of this step (**master result file**).

```
*****************************************************************
*_Start_master_timing:
*****************************************************************
Correlation method                    :       magspace (321,193)
Number of correlation windows used    :       10 of 10
Estimated translation master w.r.t. synthetic amplitude (master-dem):
  Positive offsetL: master image is to the bottom
  Positive offsetP: master image is to the right
Coarse_correlation_translation_lines  :       -1
Coarse_correlation_translation_pixels :       5
Master_azimuth_timing_error           :       0.000595265 sec.
Master_range_timing_error             :       -1.31839e-07 sec.
*****************************************************************
* End_master_timing:_NORMAL
*****************************************************************
```

In the logfile some additional information is written such as estimated offset and correlation for each window position. Estimated offsets equal to NaN are ignored during analysis.

7.3 Implementation

The master timing error is computed in three steps:

First for each window position, the (zero meaned) master amplitude window is shifted over the (zero meaned) simulated amplitude and the correlation is obtained (see equation D.24) by computing all pointwise products and dividing by the norms of the particular windows.

Second, we select the most frequently occurring line and pixel offset pair by searching for the highest frequency of window offsets above the average correlation.

Finally, we convert the obtained integer line and pixel offset to the master timing error using the following formulation

$$\text{master\_azimuth\_timing\_error} = \frac{-(\text{lines})}{\text{PRF}}, \tag{7.1}$$

$$\text{master\_range\_timing\_error} = \frac{-(\text{pixels})}{2 \cdot \text{RSR}}, \tag{7.2}$$

where PRF is the pulse repetition frequency and RSR is the one-way resampling rate in Hz.

The algorithms which are used to compute correlation are the same as the ones used in the step **COARSE_CORR**, see the related chapter for details on the magspace and magfft methods.

Chapter 8

M_OVS

In this chapter the processing of step M_OVS is described. This step can be run **optionally** to oversample the cropped data, and is done after M_CROP (and optinally after M_SIMAMP and M_TIMING).

Range oversampling has been implemented by Raffaele Nutricato, who uses an oversampling factor of 4 in range, for advanced processing in the multi-temporal analysis.

For PS type processing, factor two in both directions seems reasonable. This avoid aliasing in the spectrum of the interferogram, which implies you can interpolate correctly in the interferogram.

8.1 Input Cards

M_OVS_OUT *master_ovs.raw*
Filename of the oversampled data.

M_OVS_OUT_FORMAT *ci2*
Output file format.

M_OVS_FACT_RNG *1*
Oversampling factor of output image in range (pixels).

M_OVS_FACT_AZI *1*
Oversampling factor of output image in azimuth (lines).

M_OVS_KERNELSIZE *16*
Kernel size (sinc) used for oversampling.

Example input cards for this step:

```
 c
 c
 comment ___OVS___
  c
M_OVS_OUT         Outdata/master_ovs.raw  // output filename
M_OVS_OUT_FORMAT  ci2              // output format
                                   // image ci2 | cr4.
M_OVS_FACT_RNG    2                // range oversampling ratio
M_OVS_FACT_AZI    2                // azimuth oversampling ratio
M_OVS_KERNELSIZE  16               // interpolation kernel length
```

8.2 Output Description

The **process control flag** at the start of the **result file** is switched to 1 at successful exit.

```
oversample:               1
```

The output section in the **result file** will resemble the following.

```
*****************************************************************
*_Start_oversample:                    slave
*****************************************************************
Data_output_file:                       Outdata/slave_ovs.raw
Data_output_format:                     complex_short
First_line (w.r.t. original_image):     101
Last_line (w.r.t. original_image):      133
First_pixel (w.r.t. original_image):    991
Last_pixel (w.r.t. original_image):     1023
Multilookfactor_azimuth_direction:      1
Multilookfactor_range_direction:        0.25
Number of lines (multilooked):          33
Number of pixels (multilooked):         132
First_line (w.r.t. ovs_image):          101
Last_line (w.r.t. ovs_image):           133
First_pixel (w.r.t. ovs_image):         3961
Last_pixel (w.r.t. ovs_image):          4092
*****************************************************************
* End_oversample:_NORMAL
*****************************************************************
```

8.3 Algorithm

Based on description by Raffaele Nutricato who provided this code: In the code, look for:

```
//____RaffaeleNutricato START MODIFICATION SECTION 1

As I explained in the previous e-mail range oversampling is obtained
as convolution of the zero filled signal with a truncated sinc.  The
loading of the image is performed line by line and consequently the
oversampling is performed line by line too.

In particular given an input signal:
   Input signal: xxxx
   and an oversampling ratio of let's say 3
   I first generate a zero-filled copy of the input signal:
   zero-filled Input signal: x00x00x00x
   then I convolve the zero-filled Input signal with the interpolation
   kernel obtaining the output signal:
   Output signal: x++x++x++x++
   where +'s are the new samples.
```

Bert Kampes implemented the azimuth oversampling. In azimuth a 6 point raised cosine kernel is used. The kernel is normalized. I also normalized the range kernel (typically 16 point sinc).

Chapter 9

S_READFILES

In this chapter the processing of step S_READFILES is described. It is the same as step M_READFILES but then for the slave image. See chapter 3 (M_READFILES) for more information on this step.

9.1 Input Cards

S_IN_METHOD *ERS* | *ASAR (ENVISAT)* | *RSAT (RADARSAT)* | *ATLANTIS* | *JERS* | *ALOS* | *TSX (TERRASAR-X)*

Method selector to read ERS, ENVISAT, RADARSAT, JERS, ALOS or TERRASAR-X header. Note that both master and slave need to be acquired by the same sensor in principle. JERS simply uses ERS programs, ATLANTIS (sar processor) uses the ceos reader for RSAT, and will write this in the Product Type Specifier field. RSAT must be tested, problems may be orbit data. In later steps, the Product field is read, and the CROP step uses the appropriate function automatically (Envisat, ERS/JERS, RSAT/ATLANTIS).

S_IN_DAT *filename*

The filename of the SLC data file. This is the only file required for method ASAR (ENVISAT).

S_IN_LEA *filename*

The filename of the SLC leader file. Not used for method ASAR (ENVISAT).

S_IN_VOL *filename*

The filename of the SLC volume file. Not used for method ASAR (ENVISAT) and TSX (TERRASAR-X).

S_IN_NULL *filename*

The filename of the SLC null file. This may be a dummy name since it is not used.

Chapter 10

S_PORBITS

In this chapter the processing of step S_PORBITS is described. It is actually the same as step M_PORBITS, but then for the slave image. See chapter 4 (M_PORBITS) for more detailed information on this step.

10.1 Input Cards

S_ORBDIR *directory name*
the tar archive directory name for the Delft Orbital Data Records.

S_ORB_INTERVAL *1*
Time interval between data points. Card M_ORB_INTERVAL has the same effect. It is not possible to have a different S_ORB_INTERVAL.

S_ORB_EXTRATIME *3*
Time before first line (and after last) for extra datapoints data points. Card M_ORB_EXTRATIME has the same effect. It is not possible to have a different S_ORB_EXTRATIME.

S_ORB_DUMP *dt*
Dump interpolated t,x,y,z to ascii file slaveorbit.dat.

Chapter 11

S_CROP

In this chapter the processing of step S_CROP is described. It is the same as step M_CROP but then for the slave image. See chapter 5 (M_CROP) for more information on this step.

11.1 Input Cards

S_CROP_IN *filename*
Filename of the SLC data file.

S_CROP_OUT *slave.raw*
Filename of the raw data **output file**.

S_DBOW *linelow linehi pixellow pixelhi*
Slave output window. You can make a cutout of the image with this card. If card omitted it defaults to total image. line/pixel 1 refers to the the first line/pixel.

S_DBOW_GEO *lat_0 lon_0 height width*
Slave output window. Alternative to and overrides normal DBOW card. You can make a cutout of the image with this card. latitude of the center pixel of the desired crop, longitude (in decimal degrees, WGS84 system of orbit), then height, width in pixels. For approximately square areas, heights should be a factor 5 of width for ERS.

Chapter 12

S_OVS

In this chapter the processing of step S_OVS is described. It is the same as step M_OVS but then for the slave image. See chapter 8 (M_OVS) for more information on this step.

12.1 Input Cards

S_OVS_OUT *slave_ovs.raw*
Filename of the oversampled data.

S_OVS_OUT_FORMAT *ci2*
Output file format.

S_OVS_FACT_RNG *1*
oversampling factor of output image in range (pixels).

S_OVS_FACT_AZI *1*
oversampling factor of output image in azimuth (lines).

S_OVS_KERNELSIZE *16*
Kernel size (sinc) used for oversampling.

Chapter 13

COARSEORB

This chapter describes the processing step COARSEORB. In this step the coregistration based on the orbits of slave and master is computed with an accuracy of about 30 pixels (precise orbits). This is a fast way to get the coarse offsets. Before the FINE coregistration however the step COARSECORR has to be run, in order to get the coarse offsets within a few pixels. (FINE requires the initial estimated offset within a few pixels.)

The offset is defined in such a way that for a point P in the master with coordinates Pm(line,pixel) and the same point in the slave image with (slave system) coordinates Ps(line,pixel) it holds:

$$P_s(l,p) = P_m(l,p) + \mathrm{offset}(l,p) \tag{13.1}$$

13.1 Input Cards

There are no input cards for this step. (i.e. the parameters/orbits are read from the master and **slave result file**.)

13.2 Output Description

Since this normally is the first step that is not specific to master nor slave, a **products result file** is created. The **process control flag** at the start of this file is switched to 1 at successful exit.

```
coarse_orbits:           1
```

Example of output of this step.

```
***************************************************************
*_Start_coarse_coregistration_based_on_orbits
***************************************************************
Some info for pixel: 3037, 590 (not used):
Bperp      [m]:           36.1
Bpar       [m]:           23.9
Bh         [m]:           41.8
Bv         [m]:          -11.4
B          [m]:           43.3
alpha      [deg]:        -15.4        // baseline orientation
theta      [deg]:           18        // look angle
Height_amb [m]:          202.8
Btemp:     [days]:          -1
```

```
Estimated translation slave w.r.t. master:
Coarse_orbits_translation_lines:        236
Coarse_orbits_translation_pixels:       3
*****************************************************************
* End_coarse_orbits:_NORMAL
*****************************************************************
```

In the logfile some extra information is given, such as the number of iterations The baseline parameters are not used, but given here to make it possible to write scripts that grep these values if a lot of interferograms are processed of the same scene.

13.3 Implementation

The algorithm described in Annex D is used for the conversion between (line,pixel) coordinates to the corresponding point P on an ellipsoid. (The Doppler, range and ellipsoid equation.) This step consists of three steps basically.

1. For the center (line,pixel) of the master image, compute the position (x,y,z in system of the orbits) of the point P on an ellipsoid.
2. Based on the Doppler equation, compute the position of the slave satellite, corresponding to the point P on an ellipsoid, and compute the (line,pixel) coordinates in the slave system.
3. The difference (slave-master) between the (line,pixel) coordinates is defined as the offset.

Chapter 14

COARSECORR

In this chapter the processing of step COARSECORR is described. The offset in line (azimuth) and pixel (range) direction between master and slave is computed with an accuracy of about 1 pixel (1 offset for whole image).

The magnitude images are used; correlation is computed in the space or spectral domain.

At a number of positions (geometrically distributed or at positions read from an **input file**) in the image the correlation between master and slave is computed for different offsets. The offset with the highest correlation is the estimate for that position. The (approximate) offset between the two images is set to the offset that most occurred over the positions, so the one that is most likely. (Sometimes an estimated offset is totally unreliable, for example for a position in a sea, but the correlation is not very small. The estimated correlation at a position is likely to be biased. Therefor it would not be wise to use the offset between the two images based on the highest correlation values only, but we use this 'consistency test' instead.)

14.1 Input Cards

CC_METHOD *magfft* | *magspace*
Method selector for this step. Either perform the correlation computation on the magnitude images in he space or in the spectral domain.

CC_IN_POS *filename*
Input filename for ascii file with positions in original master system to place windows for correlation computations.

CC_NWIN *11*
Number of windows to be distributed over the total image to estimate the offset. Should be at least 5 or so because the most consistent estimate is selected. This card is ignored if CC_IN_POS is set. Only 1 large window could be used, e.g., of size 1024x1024.

CC_WINSIZE *64 64*
Size of the window in lines pixels. For method in space domain it defaults to 64 64 For method in space domain it is converted to odd numbers if necessary.

CC_ACC *32 8*
ONLY for method in space domain. Accuracy to search within for maximum correlation. For fft method it automatically equals half of the **CC_WINSIZE**.

CC_INITOFF *0 0* | *"orbit"*

Initial offset for coarse co-registration. if the word ”orbit” then the estimate of the step **COARSEORB** are read from the **products result file** and used. If there are 2 numbers then these are used.

Example input cards for this step:

```
 c
 c
 comment ___COARSE CORR (COREGISTRATION)___
 c
CC_METHOD       magfft         // default
c CC_METHOD       magspace     // (no veclib)
c CC_ACC          30 30        // (only for magspace)
CC_NWIN         21             // number of windows
CC_WINSIZE      1024 512       // size of windows
CC_INITOFF      orbit          // use result of orbits
                               //   for initial offset
c CC_INITOFF    0 0            // use this if no precise orbits
```

14.2 Output Description

At successful exit, the **process control flag** is switched to 1 in the **products result file**. If this file does not exist, it is created (I_RESFILE card):

```
coarse_correl:          1
```

The output in the (products) **result file** resembles:

```
*****************************************************************
*_Start_coarse_correlation
*****************************************************************
Estimated translation slave w.r.t. master:
Coarse_correlation_translation_lines:   241
Coarse_correlation_translation_pixels:  3
*****************************************************************
* End_coarse_correlation:_NORMAL
*****************************************************************
```

In the logfile the estimated offset is given for all windows.

14.3 Implementation

14.3.1 Method magspace

The implementation in the space domain requires an odd window size, which is automatically forced (not strictly necessary, but this made the implementation a bit easier because the center of the shifting window is defined at a pixel.) For each location the (zero meaned) slave magnitude window is shifted over the (zero mean) master window, and the correlation is computed (see equation D.24) by computing all pointwize products and dividing by the norms of the particular windows.

14.3.2 Method magfft

The implementation in the frequency domain is more or less the same as in the space domain. We only use FFT's to compute the products for the correlation (see equation D.24) in an efficient way, due to the fact that a convolution in the space domain corresponds to a multiplication in the frequency domain. Input are the zero mean magnitude images.

The cross products are obtained by computing the pointwize product of the zeropadded master x conj(slave). A block function is used to compute the norms. Note that the correlation window (the overlap) does not have a constant size with this method, but varies between winsizeL/P and .5winsizeL/P.

Chapter 15

M_FILTAZI

In this chapter the processing of step M_FILTAZI is described. This *optional* step filters the spectrum of the master in azimuth direction. The part of the spectrum that does not overlap with the spectrum of the slave is filtered out. This non overlap is due to the selection of a Doppler centroid frequency in the SAR processing, which normally is not equal for master and slave image.

This step can in general best be performed after the COARSE coregistration and before the FINE. (The coarse offset in pixel direction is used to evaluate the polynomial for the Doppler Centroid frequency.) The FINE steps can benefit a lot from this filtering (TODO add plots).

By processing the RAW data to SLC at the mean Doppler centroid frequency this step can be avoided in the InSAR processing chain. (For ESA SLC images this cannot be done obviously.)

Normally the step S_FILTAZI is performed at the same time. (requires a PROCESS S_FILTAZI card in the **input file**, see chapter 16.) However, we kept this two seperate steps to be able to only filter the slave images in a large stack (all slaves coregistered on the same master image). This has the advantage that for each interferogram of the stack not a large file is created for the master. The disadvantage of not filtering the master of course is that a small part of the spectrum of the master is not shared with the slave spectrum, yielding loss of coherence in the interferogram.

15.1 Input Cards

AF_BLOCKSIZE *1024*
Length of fft per buffer in azimuth direction. In general, the larger the better.

AF_OVERLAP *AF_BLOCKSIZE/8*
Half of the overlap between consecutive bufferes in azimuth direction. Partially the same data is used to estimate the spectrum, which might have certain advantages. However it has not been studied yet if taking an overlap is requird. Setting this card to 0 is fastest.

AF_HAMMING *0.75*
The weighting of the spectrum in azimuth direction. The filtered output spectrum is first de-weighted with the specified hamming filter, then re-weighted with a (newly centered) one. If this parameter is set to 1, no weighting is performed. For more information see 22.3.3.

AF_OUT_MASTER *master.afiltered*
Output file name for the master image.

AF_OUT_SLAVE *slave.afiltered*
Output file name for the slave image.

AF_OUT_FORMAT *cr4*
Format of outut data. Either complex real4 (cr4) or complex shorts (ci2).

An example of the **input file** (save general cards):

```
PROCESS         m_filtazi
PROCESS         s_filtazi
 c                                      //
 c                                      //
 comment ___AZIMUTH FILTERING___        //
 c                                      //
 c AF_METHOD
AF_BLOCKSIZE    1024                    // fftlength each column
AF_OVERLAP      64                      // hbs
AF_HAMMING      0.75
AF_OUT_MASTER   Outdata/1393.azifilt
AF_OUT_SLAVE    Outdata/21066.azifilt
AF_OUT_FORMAT   ci2
```

15.2 Output Description

In the process control array, the switch for azimuth filtering is turned on:

```
filt_azi:               1
```

In the **master result file** a section is added with the new file name:

```
*****************************************************************
*_Start_filt_azi:
*****************************************************************
Input_file:                             Outdata/1393.raw
Data_output_file:                       Outdata/1393.azifilt
Data_output_format:                     complex_real4
First_line (w.r.t. original_master):    1
Last_line (w.r.t. original_master):     3500
First_pixel (w.r.t. original_master):   1
Last_pixel (w.r.t. original_master):    500
*****************************************************************
* End_filt_azi:_NORMAL
*****************************************************************
```

A file (mph) is created for the master, with filtered spectrum. Figure 15.1 demonstrates the filter for 2 images:

15.3 Implementation

For each buffer of *AF_BLOCKSIZE* lines and *width* pixels do (taking care of AF_OVERLAP)

- Take 1DFFT in azimuth direction (over the columns).

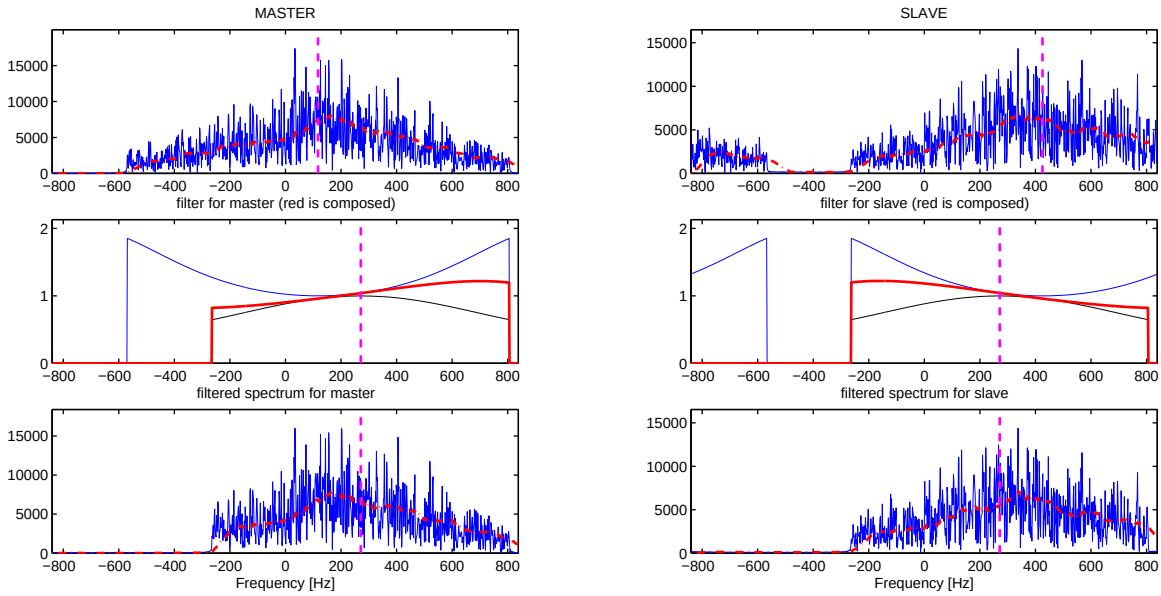

Figure 15.1: Azimuth filtering for a master (left) and slave (right) SLC image (frame 2781, orbit 1393 (master, ERS2, 27-JUL-1995) and orbit 21066 (slave, ERS1, 26-JUL-1995)). The Doppler centroid frequency for the master is $f_{DCm} = 117$ Hz (constant for all columns), for the slave $f_{DCs} = 425$ Hz, (obtained from the **result file** (read from SLC leader)). The mean Doppler centroid equals $f_{DC} = 271$ Hz. (Doppler centroid are indicated by dashed magenta lines, x axis are frequencies from [-PRF/2:PRF/2].) The azimuth spectrum was weighted with a Hamming window ($\alpha = 0.75$). (Pictures on first row, original spectra for range column 101, red dashed line is a 51 point moving average). The filtering (middle row) first de-weights by 'inverse' Hamming, centered at the image Doppler centroid, and bandlimited to the total azimuth bandwidth ($ABW = 1378$ Hz). Next a new Hamming filter is applied, centered at the mean Doppler centroid, and bandlimited to $ABW - 2 \parallel f_{DCm} - f_{DC} \parallel = 1070$ Hz. Obviously, the filter for the slave is the inverse of that of the master. The resulting spectra are shown in the bottom row. The frequencies that did not overlap are filtered out, yielding a better coherence between master and slave image. The spectrum and filters depicted here are FFT shifted for clarity.

- if the Doppler centroid frequencies do not vary per column, use the same filter for all columns, else compute the correct filter foreach column and use that. (First coarse coreg, align, then evaluate FDC polynomial.)
- Take inverse 1DFFT in azimuth direction (over the columns), yielding the output.

The azimuth spectrum is also weighted for the antenna pattern,

$$frac\sin((f_a - f_{DC})/f_{Dop})\pi((f_a - f_{DC})/f_{Dop})^2 \tag{15.1}$$

Where:
$f_{Dop} = 1505$ Hz, the Doppler bandwidth, see [Geudtner, 1996].

We did not de-weight (and re-weight) the spectrum for this. This might be visible in figure 15.1 as a slightly asymmetric spectrum, for master and slave. We are not convinced that this re-weighting can be performed, without changing the signal. However we believe that possible errors are small.

Chapter 16

S_FILTAZI

In this chapter the processing of step S_FILTAZI is described. Normally the step S_FILTAZI is performed at the same time as M_FILTAZI. (PROCESS M_FILTAZI card in **input file**.) However, we kept this two seperate steps to be able to only filter the slave images in a large stack (all slaves coregistered on the same master image). This has the advantage that for each interferogram of the stack not a large file is created for the master. The disadvantage of not filtering the master of course is that a small part of the spectrum of the master is not shared with the slave spectrum, yielding coherence loss of the interferogram.

Further information on the input/output of this step can be found in Chapter 15 (M_FILTAZI).

Chapter 17

FINE

In this chapter the processing of step FINE is described. The offset vectors to align the slave image to the master are computed with sub pixel accuracy for a number of locations in the master. Over the total image, for a large number of windows (e.g., 500, distributed by Doris or from a file with locations in the master coordinate system), the offset between master and slave is estimated by computing the correlation of the magnitude images for shifts at pixel level. Next, in a local neighborhood of the maximum (correlation at pixel level) these correlations are harmonically oversampled (interpolated, requires FFT) to find the maximum at sub pixel level. These offsets are then written to the (products) **result file**. The offset is computed in the spectral or in the space domain (which is implemented to avoid the use of FFT, but that is required later anyway, and to provide a check of the method in the spectral domain, which should be faster). The correlation is computed on the magnitude images. Though we believe this to be a good method, we would like to investigate first oversampling the images itself, and directly computing the correlation for a small number of shifts (assuming initial offsets are known within a few pixels), as we suspect that there may be an error introduced due to aliasing with the method that is implemented. (This method will be named 'oversample'.)

The actual computation of the transformation model (2d polynomial) is done by the step **COREGPM** (computation of coregistration parameters). See also [Samson, 1996].

17.1 Input Cards

FC_METHOD *magfft* | *magspace* | *oversample*
Select method for the computation. Compute cross-correlation based on magnitude images either in the space or the spectral domain. Magnitude patches are zero-meaned. Method magfft is fast, but patch size varies depending on shift. Magspace keeps constant patch-size and shifts it over the master, but is slower. Computations are done in space domain. Method oversample is best, theoretically, avoids aliasing of spectrum when magnitude is computed (using FFTs).

FC_NWIN *400*
The number of windows to be distributed over the total image. if points are read from file (FC_IN_POS), then this card is ignored.

FC_IN_POS *file name*
A ascii file with (integer line pixel pairs) coordinates in the original master coordinate system with locations where the windows should be placed. After the last coordinate there should NOT be a EOL (enter) (though Doris should ignore this). The coordinates should be within the current overlap of master and slave.

FC_WINSIZE *32 32*

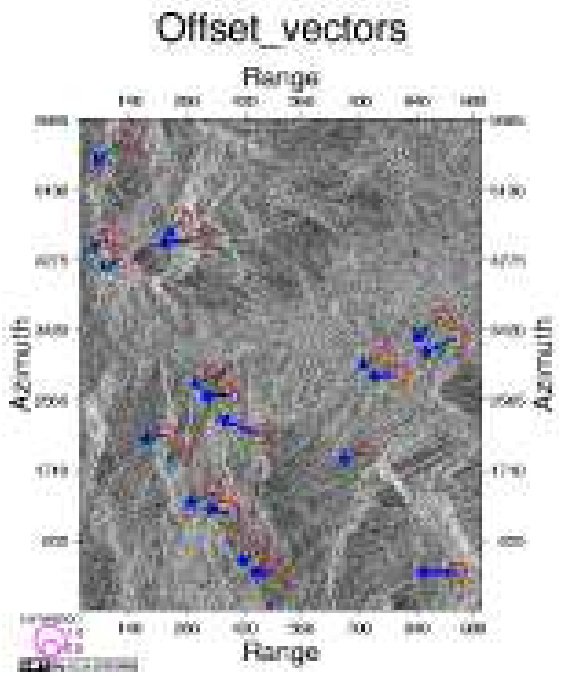

Figure 17.1: Plot produced by the command 'plotoffsets interferogram.out 11 6000 21 1000 0.6 Outdata/1393.raw' (keycard FC_PLOT 0.6 BG). The magnitude is plotted in the background. Correlation is indicated by the size of the circles, estimates with a correlation below 0.6 are filtered out.

The size of the correlation window. Recommended is 64 64

FC_ACC *4 4*

The search accuracy for the maximum correlation. Adviced is 8 8. (total search area is from -Acc to +Acc). for FFT methods this must be a power of 2. In the logfile after step COARSECORR the variation of the initial offsets w.r.t. the estimated values can be seen. If this variation is larger than 1 (1 is normal for ERS1/2 SLC images) then one should select a bigger window and a larger search accuracy.

FC_INITOFF *0 0* | *COARSECORR*

The initial offset between master and slave. "COARSECORR" indicates that the results of the step **COARSECORR** are used.

FC_OSFACTOR *16*

The oversampling factor for the harmonic interpolation of the correlation. Recommended is 32 to co-register the images within a tenth of a pixel.

FC_PLOT *threshold=0.4 NOBG* | *BG*

Call gmt script plotoffset to plot results and to view with gv. (An example of a plot is given above.) This script gets the section with estimated fine offsets from the interferogram **result file**. The argument threshold filters out estimates with a correlation below this threshold. The second argument (BG or NOBG) selects a call to cpxfiddle to generate a magnitude background, while BG does call cpxfiddle. See the script plotoffsets and the c program cpxfiddle for more information. cpxfiddle can be downloaded from Doris internet pages. The command is echoed to stdout as INFO, which can be repeated outside Doris. Before running the step COREGPM to estimate a transformation model, it is very convenient to view a number of offset vectors above a correlation threshold to select the appropriate value for the card CPM_THRESHOLD. Actually, a background call is made to the script 'plotoffsets' (something like: 'plotoffsets interferogram.out 11 6000 21 1000 0.6'). This command can be given from the prompt as well, for different values of the threshold.

With a command like:

```
awk 'BEGIN{for (i=100;i<25200;i=i+500) \
{for (j=750;j<5400;j=j+200) \
{printf "%i %i \n",i,j}} exit}'
```

the file for FC_IN_POS can be easily generated for a grid of locations.

Example input cards for this step:

```
 c
 c
 comment ___FINE COREGISTRATION___
 c
FC_METHOD       oversample       //
 c FC_METHOD       magfft        //
 c FC_METHOD     magspace        //
FC_NWIN         101              // number of windows
FC_WINSIZE      64 64            // size of windows
FC_ACC          8 8              // search window, 2^n
FC_INITOFF      coarsecorr       // use result of coarse to compute first
FC_OSFACTOR     32               // oversampling factor
```

17.2 Output Description

The **process control flag** at the start of the **products result file** is switched to 1 at successful exit.

```
fine_coreg:             1
```

Example of output of this step (**products result file**).

```
*****************************************************************
*_Start_fine_coreg
*****************************************************************
Oversampling factor:            16
Number_of_correlation_windows:  100
Number  posL    posP    offsetL offsetP correlation
   0    27    27    241.06      3.12   0.16
   1    27   237    241.12      3.06   0.18
   2    27   447    241.12      3.31   0.44

[SKIP] [SKIP]

  99  4733   761    241.06      3.44   0.15
```

```
 100  4733   971    241.12      3.44   0.14

*****************************************************************
* End_fine_coreg:_NORMAL
*****************************************************************
```

In the logfile addition information is given.

17.3 Implementation

The current names for the master and slave image are read from the **result files**, crop section. Here also the dimensions of the files are read. This can be checked with the debug version of Doris. Doris can be tricked to coregister other complex files, e.g., complex interferograms for 4 pass differential interferometry, by substitution the right parameters in that section.

17.3.1 magspace

The computations are similar to the COARSECORR magspace method.

17.3.2 oversample

See source code.

17.3.3 magfft

The correlation is computed at pixel level, similar to step COARSECORR. These computations are described in that chapter. (That step still has to be performed because the FINE step requires accurate initial estimates of offsets. The AccL and AccP cards define the size of the searchwindow (2*AccL x 2*AccP) around the initial offsets to interpolate a maximum.)

The oversampling is done as follows:

1. Transformation to spectral domain of searchwindow with correlation values at pixel level.
2. Padd with zeros, half the last term.
3. Inverse transform.
4. Find maximum in space domain, this corresponds to estimated offsetvector.

Note that this way of computing is exactly the same if you first interpolate the signal and compute all correlations and find the maximum, or that you first compute at pixel level and interpolate the correlation values.

Chapter 18

RELTIMING

This chapter describes the processing step RELTIMING. The relative timing error between the master and slave is computed using the (precise) orbits and the result of the fine coregistration. Therefore, this step can only be run after the steps **COARSE_ORBIT** and **FINE**. The timing error in azimuth as well as in range direction is estimated.

This step is of influence for the coregistration using a DEM (**DEMASSIST** step). For this coregistration the master and slave images should be aligned as good as possible with the DEM. The master is aligned with respect to the DEM using a simulated amplitude image (**M_TIMING** step). This could also have been done for the slave image, however, we have chosen to use the master-slave fine coregistration because this is probably more precise for the relative timing. Therefore, an error in the master timing will propagate to the slave timing, however, the relative DEM position is consistent.

18.1 Input Cards

RTE_THRESHOLD *0.4*
: Threshold for correlation value to use estimated offset of step **FINE** in estimation of the correlation based offset. This depends on the size of the window during FINE. Estimated coherence using small windows are more biased towards 1.0, so a higher threshold is better. For window size 64 64 a threshold 0.2 seems OK. The plotoffsets script can be used from the prompt to figure out a good threshold value.

RTE_MAXITER *10000*
: Maximum number of iterations in least squares adjustment and testing procedure. The least squares adjustment and testing is repeated, until all tests are accepted, or the maximum number of iterations is reached. If the maximum number of iterations is reached before acceptance of all tests, this is an indication that the fine coregistration failed or is of bad quality. In this case, either run the **FINE** step again (obviously with different settings) or increase the RTE_K_ALPHA card (to accept a lower quality coregistration).

RTE_K_ALPHA *1.97*
: Critical value of outlier detection. A higher value accepts more outliers. This value can be found as the sqrt of the normal distribution. Typical values are 1.97 and 3.29.

Example input cards for this step:

```
c
c
```

```
 comment ___RELATIVE TIMING ERROR___
 c
RTE_THRESHOLD   0.4
RTE_MAXITER     10000
RTE_KALPHA      1.97
```

18.2 Output Description

The **process control flag** at the start of the **products result file** is switched to 1 at successful exit.

```
timing_error:              1
```

Example of output of this step (**products result file**).

```
*****************************************************************
*_Start_timing_error:
*****************************************************************
Orbit_azimuth_offset (master-slave):            197 lines.
Orbit_range_offset (master-slave):              18 pixels.
Estimated_azimuth_offset (master-slave): 193.787 lines.
Estimated_range_offset (master-slave): 17.5939 pixels.
Estimated_azimuth_timing_error_lines (master-slave): 3 lines.
Estimated_range_timing_error_pixels (master-slave): 0 pixels.
Estimated_azimuth_timing_error_sec (master-slave): 0.00178588 sec.
Estimated_range_timing_error_sec (master-slave): 0 sec.
*****************************************************************
* End_timing_error:_NORMAL
*****************************************************************
```

In the logfile some additional information is written.

18.3 Implementation

The observation equations are given by the zero-degree polynomial model (y = A x):

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} \alpha_{l=0,p=0} \end{bmatrix} \tag{18.1}$$

Where:
y contains the observed offsets in a certain direction.
α_{lp} denotes the unknown coefficients of the polynomial.

The least squares parameter solution is given by:

$$A^T Q_y^{-1} y = A^T Q_y^{-1} A x = N x \tag{18.2}$$

Where:
Q_y^{-1} is the (diagonal) covariance matrix of the observations.

The coefficients are estimated by factorization of the matrix N.

The inverse of matrix N is also computed to check the solution (stability) and to compute some statistics.

A check number is given ($\max(\mathrm{abs}(NN^{-1} - I))$) that gives a hint on the stability of the solution.

Chapter 19

DEMASSIST

In this chapter the processing of step **DEMASSIST** is described. Here the slave image is coregistered to the master image based on a DEM. For each pixel of the master image the corresponding (real valued) coordinate in the slave image is computed.

For this step a DEM is required, e.g., obtained by the SRTM mission. The Doris distribution contains the utility *construct_dem.sh* do download and prepare SRTM data (see Section C.2.12). The utility also outputs a figure of the final result (.ps format) and the lines needed for the Doris input file.

The DEM-assisted coregistration is not dependent on the correlation between the master and the slave image. Coregistration errors due to bad distribution or lack of useful correlation windows in the conventional method are therefore prevented. Furthermore, the coregistration improves in case of large baselines and strong topography. The improvement is especially significant in case of X-band data. An analysis of the performance of the implemented DEM assisted coregistration for various sensors is described in [Arikan et al., 2008, Nitti et al., 2008].

19.1 Input Cards

DAC_IN_DEM *filename*
filename of input DEM (gtopo30). File is assumed to be stored in a raster. Major row order. from North to South, line-by-line. See also internet links at Doris home page for available DEMs.

DAC_IN_FORMAT *I2* | *I2_BIGENDIAN* | *R4* | *R8*
format of input DEM on file (signed short for gtopo30, or real4, or real8; input matrix is raw binary data w/o header, endianness of host platform is assumed, except for I2_BIGENDIAN).

DAC_IN_SIZE *6000 4800*
Number of rows and columns of input DEM file. Default is set to tile w020n90.DEM.

DAC_IN_DELTA *0.0083333333333333333* [*deltalon*]
Grid spacing of input DEM in decimal degrees, latitude longitude. Default is equal gridspacing, default set to tile w020n90.DEM.

DAC_IN_UL *89.995833333333333 -19.995833333333333333333*
Coordinates of UL (upperleft) corner in decimal degrees, latitude [-90, 90] longitude [-180, 180]. Default is set to tile w020n90.DEM. It is interpreted as max-latitude, min-longitude in source.

DAC_IN_NODATA *-9999*
Identifier to ignore data in input DEM with this value. Default is set to tile w020n90.DEM.

DAC_OUT_DEM *filename*
Request optional debug output to float file of input DEM per buffer, cut to the interferogram window. Info on these files is written as DEBUG.

DAC_OUT_DEMI *filename*
Request optional debug output to float file of interpolated input DEM, cut to the interferogram window. Info on these files is written as DEBUG.

DAC_OUT_DEM_LP *demheight_lp.raw*
Filename of output DEM height in radar coordinates.

Example input section:

```
c       ___           ___
comment ___DEMASSIST___
c
 DAC_IN_DEM        final_wanaF2835.dem
 DAC_IN_FORMAT     r4
 DAC_IN_SIZE       3601 3601
 DAC_IN_DELTA      0.000833333 0.000833333
 DAC_IN_UL         40  27                  // the center cn of UL corner p$
 DAC_IN_NODATA     -32768
 DAC_OUT_DEM      dem_dac.raw
c DAC_OUT_DEMI   demi_dac.raw
c DAC_OUT_DEM_LP   demLP_dac.raw
```

19.2 Output Description

At successful exit the **process control flag** is switched on:

```
demassist:            1
```

The output (in the **products result file**) looks like:

```
*********************************************************************
*_Start_dem_assist:
*********************************************************************
DEM source file:                        final_wanaF2835.dem
Min. of input DEM:                      92
Max. of input DEM:                      1815
First_line (w.r.t. original_master):    3053
Last_line (w.r.t. original_master):     8052
First_pixel (w.r.t. original_master):   1714
Last_pixel (w.r.t. original_master):    2713
Number of lines:                5000
Number of pixels:               1000
Deltaline_slave00_dem:          -194.341
Deltapixel_slave00_dem:         -17.7004
```

```
Deltaline_slave0N_dem:          -194.082
Deltapixel_slave0N_dem:         -18.8867
Deltaline_slaveN0_dem:          -194.479
Deltapixel_slaveN0_dem:         -17.8064
Deltaline_slaveNN_dem:          -194.232
Deltapixel_slaveNN_dem:         -18.9409
*****************************************************************
* End_dem_assist:_NORMAL
*****************************************************************
```

The output in the logfile is more verbose, specifying the results of the intermediate steps. Also go over the standard out in case of problems, with the SCREEN set to DEBUG level.

19.3 Implementation

The DEM-assisted coregistration is estimated in two steps:

First, the DEM is radarcoded to the coordinate systems of the master and slave image. Per DEM point the master coordinate and the offset between master and slave is saved to a file. Both the master coordinates and offsets are real valued.

Second, the offsets are interpolated to the integer grid of master coordinates. A linear interpolation based on a Delaunay triangulation is used. The software package *Triangle* for the Delaunay triangulation is kindly made available by Jonathan Shewchuk [Shewchuk, 1996, Shewchuk, 2002], see also `http://www.cs.cmu.edu/~quake/triangle.html`.

The finally optained master-slave offsets per master pixel are saved to a file and used in the **RESAMPLE** step to resample the slave image to the master geometry.

Chapter 20

COREGPM

This chapter describes the processing step COREGPM is described (coregistration parameters, computation of a polynomial that models the alignment of slave on master).

Based on the estimated offsets computed in step **FINE**, a 2d-polynomial model of certain degree of the coregistration is computed. When the step **DEMASSIST** is applied, the polynomial is estimated through the *residuals* between the **FINE** and **DEMASSIST** results. This appears necessary, because we experienced a remaining trend in the residuals. In this case, a 1-degree polynomial seems sufficient. A least squares solution is used, solution by cholesky decomposition of the normal matrix. Data may be excluded a priori by setting a threshold value for the correlation. Data can also be excluded by editing the **products result file** and artificially decrease the correlation for a certain offset window.

After the computations, the residuals between the estimated model and the 'observed' offsets are plotted with the csh-script *plotcpm*. These plots are useful to iteratively come to a good transformation model (changing CPM_THRESHOLD or CPM_DEGREE for each iteration, or identify and remove some estimated offsets (the 'observations', blunders) by setting their correlation to 0.00001 in the output section of the **FINE** processing step.

Also the observations itself and some statistics are plotted (w-tests, a large value indicates an unreliable estimate).

The script can be adapted to your own wishes, it simply calls GMT (see [Wessel and Smith, 1998]) based on the ascii data file CPM_DATA.

This step is important, since the interferogram is sensestive to mis alignments of slave on master. Therefore, we always took a very cautious approach. However, that meant running this step, editing the **result file**, running again, etc. which got quite cumbersome. To reduce the manual effort, we introduced a card CPM_MAXITER that performs a number of iterations automatically. It also should remove no more windows than necessary for a good fit. I have experimented with values like 20 for this card. (having say 600 windows after step FINE). If you want to approach that after each computation you want to have full control what to do, simply set this card to 0.

The first run of coregpm for the area of Fig. 23.1 is shown in Figures 20.1, 20.2, and 20.3. We have used a polynomial of degree 1, and a threshold of 0.4 here.

The second run of coregpm is shown in Figures 20.4, 20.5, and 20.6. In the **products result file** the outliers are artificially set to 0 correlation (thus being below the threshold), to exclude them from the least squares estimation. After this run we continued with the resampling.

Degree d=1 is enough to account for most effects in normal images. The 2d-polynomial has the form:

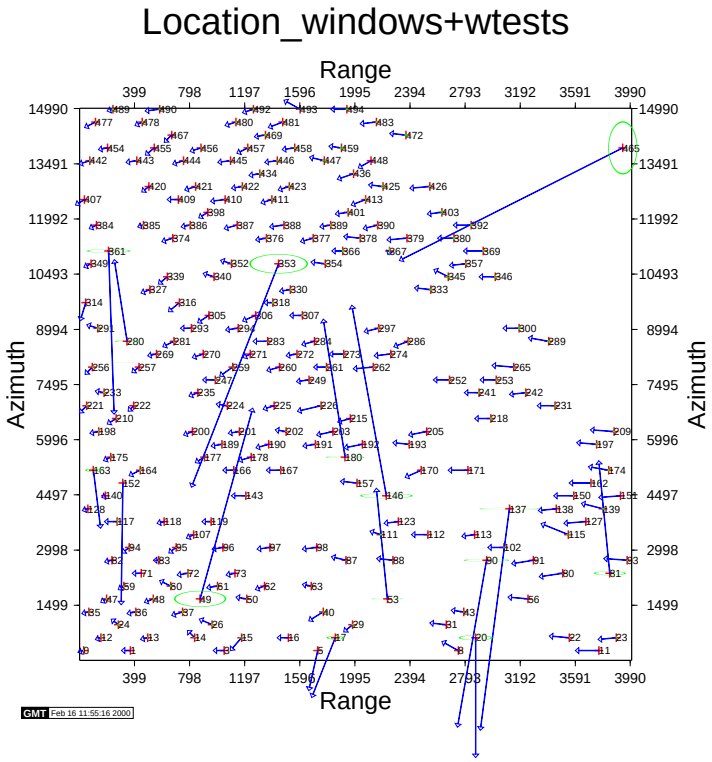

Figure 20.1: Plot produced by 'plotcpm' for the first run. The estimated offsets are plotted here (normalized), together with a (90 degrees rotated) w test as ellipses.

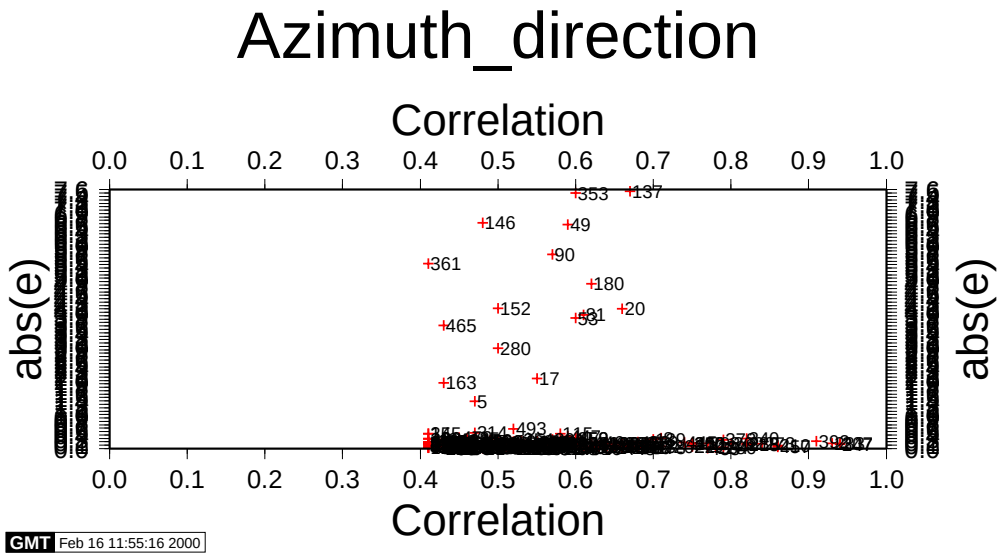

Figure 20.2: Plot produced by 'plotcpm' for the first run. The absolute error (estimated offsets minus observed offsets) are plotted for azimuth direction.

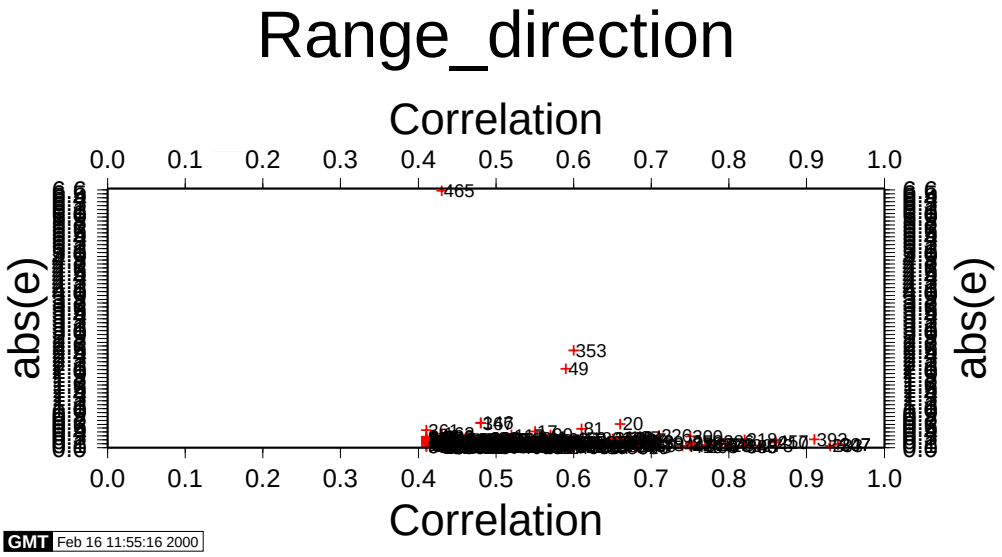

Figure 20.3: Plot produced by 'plotcpm' for the first run. The absolute error (estimated offsets minus observed offsets) are plotted for the range direction.

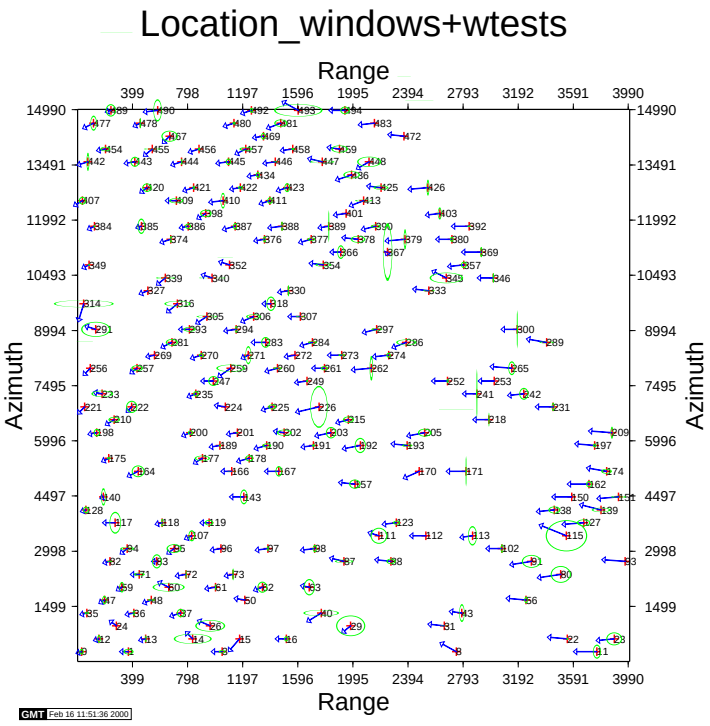

Figure 20.4: Plot produced by 'plotcpm' for the second run. The estimated offsets are plotted here (normalized), together with a (90 degrees rotated) w test as ellipses.

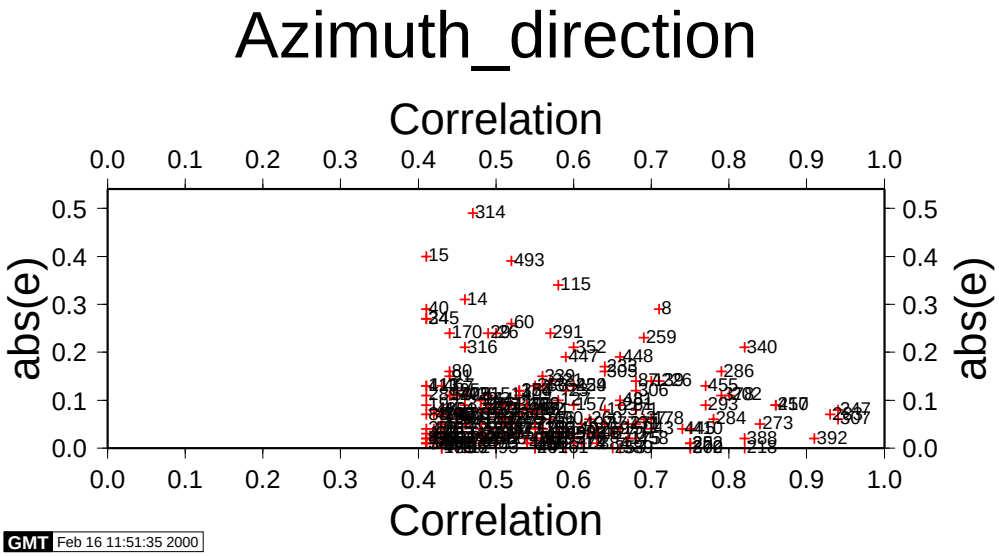

Figure 20.5: Plot produced by 'plotcpm' for the second run. The absolute error (estimated offsets minus observed offsets) are plotted for azimuth direction.

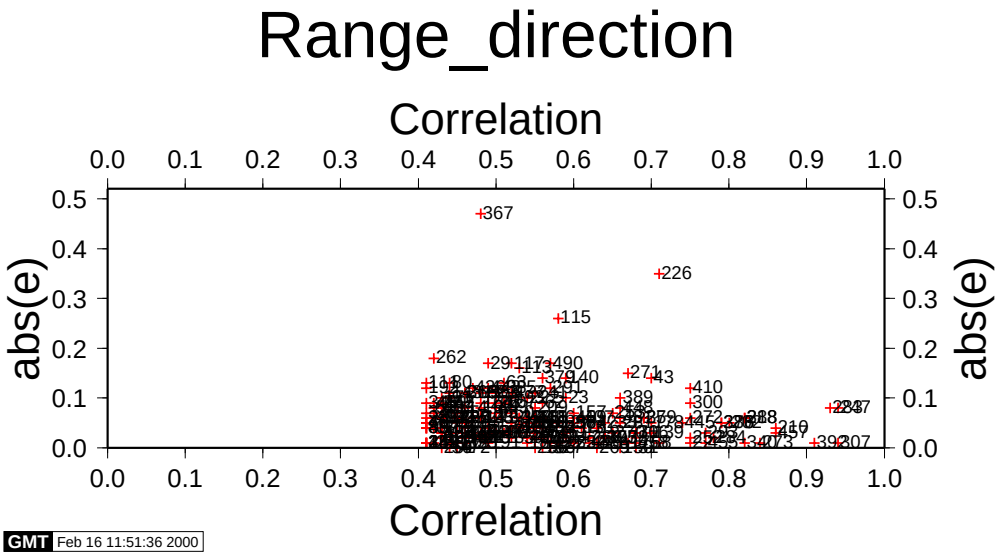

Figure 20.6: Plot produced by 'plotcpm' for the second run. The absolute error (estimated offsets minus observed offsets) are plotted for the range direction.

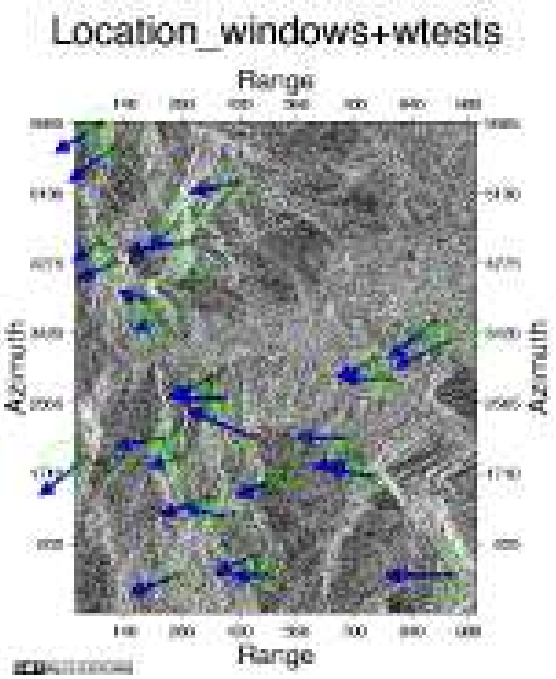

Figure 20.7: Plot produced by 'plotcpm'. The magnitude is plotted in the background.

$$f(x,y) = \sum_{i=0}^{d} \sum_{j=0}^{i} \alpha_{i-j,j} x^{i-j} y^{j} \tag{20.1}$$

Perhaps one might do the resampling with a lower quality polynomial, and thereafter do the fine coregistration (initial offsets 0,0) and this step (yielding new coefficients). The polynomial coefficient can then be added (?) to form a new model, with which the slave can again be resampled. This has not been tested.

20.1 Input Cards

CPM_THRESHOLD *0.4*
Threshold for correlation value to use estimated offset of step **FINE** in estimation of polynomial coefficients. This depends on the size of the window during FINE. Estimated coherence using small windows are more biased towards 1.0, so a higher threshold is better. For window size 64 64 a threshold 0.2 seems OK. The plotoffsets script can be used from the prompt to figure out a good threshold value.

CPM_DEGREE *1*
Degree of 2d-polynomial. See annex for definition of degree. Degree 2 is adviced.

CPM_DUMP *OFF* | *ON*
Dump computed model to files in float format. Filename for azimuth model is offsetazi_#l_#p.r4 (where number of lines,pixels are substituted). Filename for range is similar. Content of file is evaluated model in master system. via INFO the dimensions are also echoed.

CPM_PLOT *NOBG* | *BG*

Call gmt script plotcpm to plot results and to view with gv. (An example of the plots is given above.) The argument NOBG prevents a call to cpxfiddle to generate a magnitude background, while BG does call cpxfiddle. See the script plotcpm and the c program cpxfiddle for more information. cpxfiddle can be downloaded from Doris internet pages. The command is echoed to stdout as INFO, which can be repeated outside Doris.

CPM_WEIGHT *BAMLER* | *NONE* | *LINEAR* | *QUADRATIC*

Experimental card. Weight estimated offsets (observations) based on correlation in least squares solution. weighting option Bamler was added in v3.16 and made the default (recommended). The theoretical precision of shift estimation using coherent patches is the basis of this weighting option.

CPM_MAXITER *10*

Number of outlier to remove automatically based on outlier test. The least squares adjustment is repeated, until all tests are accepted, or the max. number of iterations is reached.

CPM_K_ALPHA *1.97*

Critical value of outlier detection. A higher value accepts more outliers. This value can be found as the sqrt of normal distribution. if you want a level of significance for the outlier test of 0.05, then look the value up under a half sided test.

Example input cards for this step:

```
 c
 c
 comment ___COMPUTE COREGISTRATION PARAMETERS___
 c
CPM_THRESHOLD   0.4
CPM_DEGREE      2
CPM_WEIGHT      linear                          // none
c CPM_WEIGHT      quadratic                          // none
CPM_MAXITER     20
CPM_PLOT        NOBG
```

20.2 Output Description

The plots are made if **CPM_NOPLOT** is not set. The plotcpm uses a file CPM_DATA which is created in the working directory containing the data to be plotted.

The **process control flag** at the start of the **products result file** is switched to 1 at successful exit.

```
comp_coreg:             1
```

Example of output of this step (**products result file**).

```
*****************************************************************
*_Start_coregpm:
*****************************************************************
Degree_cpm:     1
Estimated_coefficientsL:
 2.41088165e+02  0 0
-1.48768713e-05  1 0
-1.75315145e-05  0 1

Estimated_coefficientsP:
```

```
 3.11544442e+00  0 0
 7.39316101e-06  1 0
 1.91161205e-04  0 1
*****************************************************************
* End_coregpm:_NORMAL
*****************************************************************
```

In the logfile some additional statistical information is written. The standard deviation of the estimates and the residuals after the least squares adjustment.

An ascii file CPM_Data is created with some information for the plotcpm. An example is shown below:

```
File: CPM_Data
This file contains information on the least squares
 estimation of the coregistration parameters.
This info is used in the plotting scripts.
There are 10 columns containing:
Window number, position L, position P,
 offsetL (observation), offsetP (observation), correlation,
 estimated errorL, errorP, w-test statistics for L, P.
win posL posP      offL    offP  corr      eL     eP  wtstL  wtstP
-------------------------------------------------------------
0   268    30   -241.06   -3.19   0.42   0.08   0.19  81.23 200.79
1   268   369   -241.00   -3.31   0.55   0.01   0.24  15.73 249.91
3   268  1048   -241.00   -3.38   0.46   0.01   0.16  11.35 170.22
5   268  1726   -242.38   -3.31   0.47   1.39   0.05 1443.57 53.66
8   268  2744   -240.69   -3.56   0.71   0.31   0.02 323.20  19.65

[SKIP][SKIP]

492 14974  1260 -241.12   -3.38   0.41   0.14   0.18 150.86 184.45
493 14974  1599 -240.69   -3.56   0.52   0.58   0.07 601.50  72.21
494 14974  1938 -241.00   -3.56   0.42   0.27   0.14 280.63 147.84
```

20.3 Implementation

The observation equations are given by the polynomial model (y = A x):

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & l_1 & p_1 & l_1^2 & \cdots & p_1^d \\ 1 & l_2 & p_2 & l_2^2 & \cdots & p_2^d \\ \vdots & & & & & \\ 1 & l_N & p_N & l_N^2 & \cdots & p_N^d \end{bmatrix} \begin{bmatrix} \alpha_{l=0,p=0} \\ \alpha_{10} \\ \alpha_{01} \\ \alpha_{20} \\ \vdots \\ \alpha_{0d} \end{bmatrix} \tag{20.2}$$

Where:
y contains the observed offsets in a certain direction.
l_i denotes the location (line number) of the observed offsets in a certain direction.
p_i denotes the location (pixel number) of the observed offsets in a certain direction.
α_{lp} denotes the unknown coefficients of the polynomial.

The data is rescaled (to the interval [-2, 2], see Annex D) so the normalmatrix is rescaled. otherwise there could occur very high values for, e.g., $l^d = 25000^5$. The least squares parameter solution is given by:

$$A^T Q_y^{-1} y = A^T Q_y^{-1} A x = N x \tag{20.3}$$

Where:
Q_y^{-1} is the (diagonal) covariance matrix of the observations. this matrix can be equal to identity or to the correlation values in version 1. (CPM_WEIGHT card).

The coefficients are estimated by factorization of the matrix N.

The inverse of matrix N is also computed to check the solution (stability) and to compute some statistics.

A check number is given ($\max(\mathrm{abs}(NN^{-1} - I))$) that gives a hint on the stability of the solution.

Chapter 21

RESAMPLE

In this chapter the resampling, or interpolation, of the slave image on the master grid is described. The slave image is resampled (reconstruction of original signal from the samples by correlation with interpolation kernels in space domain) accordingly to the transformation model from the step **COREG_PM** and optionally **DEMASSIST**. This model states with sub-pixel accuracy which points of the slave correspond to the master grid.

Note: This step may be fairly time consuming.

The spectrum in azimuth can be centered at zero before resampling, and shifted back to its original Doppler centroid frequency afterwards. This is required since the spectrum of the kernel function is centered at zero. See also [Geudtner, 1996]. This shifting has been implemented in release 3.0, but not in prior versions.

The polynomial described by the strings in the **slave result file** (Xtrack_f_DC_constant, etc.) is used. If the spectrum should be shifted, use the card RS_SHIFTAZI.

New in v3.4 is that the azimuth kernel is shifted to the Doppler centroid, not as before the dataspectrum to zero and back. This is made default.

To assess the quality of the resampling, the resampled slave image can again be coregistered (step FINE) onto the master. This should yield offset vectors that are normally distributed with zero mean. The slave image could also be resampled in steps, first resampling it by a first degree model, then again with a higher degree model.

21.1 Input Cards

RS_METHOD *RECT* | *TRI* | *CC4P* | *CC6P* | *TS6P* | *TS8P* | *TS16P* | *KNAB6* | *KNAB8* | *KNAB10* | *KNAB16* | *RC6P* | *RC12P*

Select kernel for interpolation. a simple step function (nearest neighbor), or a linear interpolation (tri), or cubic convolution kernel (4 or 6 point), or a knab sampling window using a default data oversampling factor, or a Raised Cosine (best) kernel (6 or 12 points), or a truncated sinc (6, 8 or 16 point) can be used.

RS_OUT_FILE *s_resampled.raw*

Output filename of resampled slave, cannot be equal to the **input file** name.

RS_OUT_FORMAT *CR4* | *CI2*

Output format of resampled slave, complex_real4 or complex_short (same as SLC input format, this causes an error of maximum about 1 percent (?)).

RS_DBOW *linelo linehi pixello pixelhi*

Data base output window (in master coordinate system) for slave to make a cutout. If card is omitted it defaults to the overlap between master and slave (and corrected for half the kernel size where no interpolation is possible). For stacking of interferograms on top of each other, use the coordinates of the master after cropping. In this way all interferograms are automatically aligned. If the slave image is smaller than the window, the pixels are set to 0.

RS_DBOW_GEO *lat_0 lon_0 height width*

Output window specified in latitude, longitude (decimal degrees, WGS84). Alternative to and overrides normal RS_DBOW card. The latitude and longitude of the central pixel of the desired crop are specified, together with the height, width in pixels. This card is especially useful in combination with the M_DBOW_GEO and S_DBOW_GEO cards (see Chapters 5 and 11).

RS_SHIFTAZI [*ON* | *OFF*]

If ON, then it is accounted for non centered azimuth spectrum of the data. The azimuth interpolation kernel is shifted to the Doppler center frequency before resampling. If the fDC is small, users could switch this card to OFF.

Example input cards for this step:

```
 c
 c
 comment ___RESAMPLING SLAVE___
 c
 c RS_METHOD      cc4p
RS_METHOD       cc6p
 c RS_METHOD      ts6p
 c RS_METHOD      ts8p
 c RS_METHOD      ts16p
RS_OUT_FILE     Output/01393.resampled
RS_OUT_FORMAT   ci2
c RS_DBOW         1001 2105 501 700
RS_DBOW_GEO      52.123 5.732 5000 1000
```

21.2 Output Description

The **process control flag** at the start of the **slave result file** is switched to 1 at successful exit.

```
resample:               1
```

Example of output of this step (in **slave result file**).

```
*****************************************************************
*_Start_resample
*****************************************************************
Data_output_file:                          Output/01393.resampled
Data_output_format:                        complex_short
Interpolation kernal:                      6 point cubic convolution.
First_line (w.r.t. original_master):       1001
Last_line (w.r.t. original_master):        2105
First_pixel (w.r.t. original_master):      501
Last_pixel (w.r.t. original_master):       700
*****************************************************************
* End_resample:_NORMAL
*****************************************************************
```

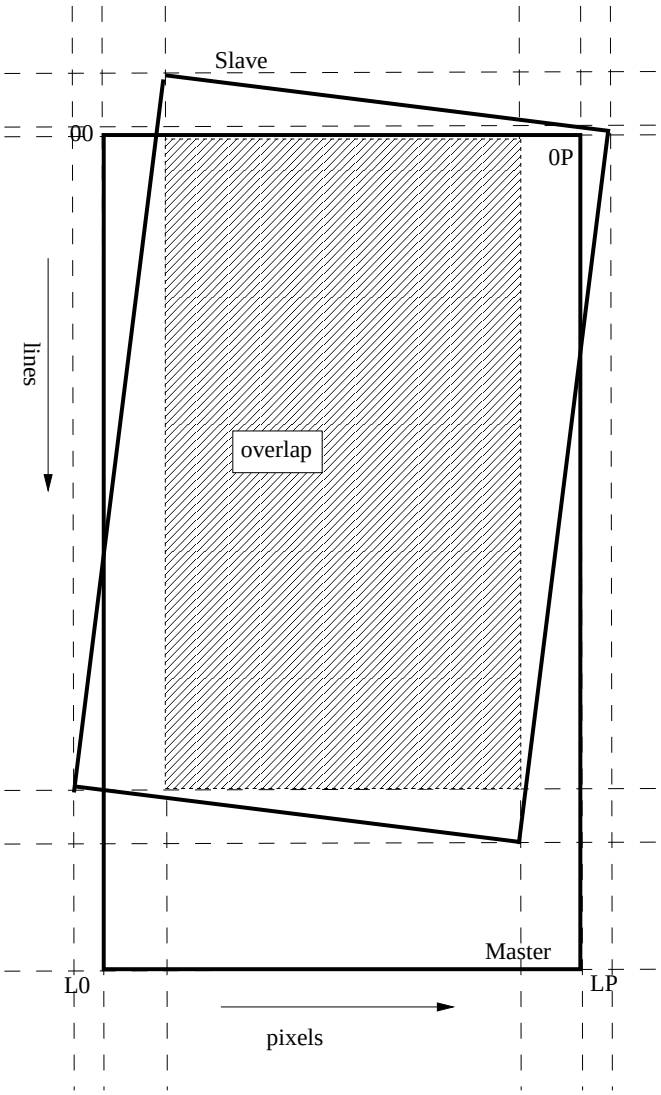

Figure 21.1: Definition of the overlap between master and slave.

Note that the line and pixel numbers are given in the master coordinate system, because the slave is interpolated to that grid now.

21.3 Implementation

The overlap between slave and master is computed as indicated in figure 21.1.

Interpolation is done with a kernel function such as, e.g., a truncated sinc function. First a look-up table is computed for the selected interpolation kernel. This table evaluates the kernel every 1/INTERVAL = 0.05 positions, which should be accurate enough.

Interpolation is independent for azimuth and range direction. For all points in the overlap between master and slave, the co-registration polynomial is evaluated.

If the total image does not fit in the memory, processing is done in buffers.

The azimuth spectrum of the complex SLC data is not centered around zero in general. The location of the peak in the spectrum is given by a polynomial in the header file.

For a correct interpolation, either the spectrum of the data has to be shifted to zero, or the interpolation kernel. We shift the kernel in azimuth. In range the spectrum is centered (for satellite data).

```
A simple derivation shows how the kernel should be shifted.
Suppose we have a signal:

s(x)=exp(i*2pi*x*fdc/prf),

and we want to interpolate this signal at xi=5.1 with a triangular
kernel k(x_0), such that the interpolated signal

s_i(xi)=sum(k(x_0).*s(x_1)).

This means with x_0=[-0.1,0.9] and x_1=[5,6] (as implemented in
Doris), the kernel is formed as

k(x_0)=triangle(x_0) = [0.9,0.1]

and we shift this with the MINUS sign by multiplication with
t(x_0)=exp(-i*2p*x_0*fdc/prf).

Then the interpolated value at 5.1 equals:

s_i(5.1) = k(-0.1)*t(-0.1)*s(5) + k(0.9)*t(0.9)*s(6)
         = 0.9*exp(-i*2pi*-0.1*fdc/prf)*exp(i*2pi*5*fdc/prf) +
           0.1*exp(-i*2pi*0.9*fdc/prf)*exp(i*2pi*6*fdc/prf)
         = exp(i*2pi*5.1*fdc/prf)
   = s(5.1);// perfect interpolation!

On the contrary, when we use the PLUS, it follows that

s_i(5.1) = 0.9*exp(i*2pi*-0.1*fdc/prf)*exp(i*2pi*5*fdc/prf) +
           0.1*exp(i*2pi*0.9*fdc/prf)*exp(i*2pi*6*fdc/prf)
         = 0.9*exp(i*2pi*4.9*fdc/prf) + 0.1*exp(i*2pi*6.9*fdc/prf)
           NE s(5.1);// wrong sign used!
```

See also [Hanssen and Bamler, 1999].

21.3.1 Output formats

Computations are done in complex float. Casting these values to complex short format introduces an error. Of course, the main advantage is a factor 2 reduction in the size of the **output file**.

Now an example follows for the error in amplitude and phase for a complex value of about (100,100). If the actual interpolated complex value equals (100.5,100.5), then the error in the magnitude approximately is

$$e_m \approx 100(\sqrt{(}100^2 + 100^2) - \sqrt{(}100.5^2 + 100.5^2))/\sqrt{(}100.5^2 + 100.5^2) = 0.5\% \tag{21.1}$$

If the actual value did equal (100.5,100.0), then the error in the phase is approximately

$$e_p \approx 100(\arctan(100/100) - \arctan(100.5/100))/\arctan(100.5/100) = 0.3\% \tag{21.2}$$

These are worst case scenarios. If the complex value is larger, then the relative error decreases. Note that the maximum for a signed short integer is $2^{15} = 32768$.

21.3.2 Interpolation Kernels

In this section the available kernels are defined. See also [Hanssen and Bamler, 1999]. The KNAB interpolation kernel is described in a IEEE letter of 2003. The Raised Cosine interpolation kernel is described in a article in J. Of electromagnetic waves, 2005. Cho et al.

$$sinc(x) = \frac{\sin \pi x}{\pi x} \tag{21.3}$$

$$rect(x) = \begin{cases} 0 & |x| > 0.5 \\ 0.5 & |x| = 0.5 \\ 1 & |x| < 0.5 \end{cases} \tag{21.4}$$

$$i(x) = tri(x) = \begin{cases} 0 & |x| > 1 \\ 1 - |x| & |x| < 1 \end{cases} \tag{21.5}$$

($\alpha = -1$)

$$i(x) = \begin{cases} (\alpha+2)|x|^3 - (\alpha+3)|x|^2 + 1 & 0 \le |x| < 1 \\ \alpha|x|^3 - 5\alpha|x|^2 + 8\alpha|x| - 4\alpha & 1 \le |x| < 2 \\ 0 & 2 \le |x| \end{cases} \tag{21.6}$$

($\alpha = -.5; \beta = .5$):

$$i(x) = \begin{cases} (\alpha-\beta+2)|x|^3 - (\alpha-\beta+3)|x|^2 + 1 & 0 \le |x| < 1 \\ \alpha|x|^3 - (5\alpha-\beta)|x|^2 + (8\alpha-3\beta)|x| - (4\alpha-2\beta) & 1 \le |x| < 2 \\ \beta|x|^3 - (8\beta)|x|^2 + (21\beta)|x| - (18\beta) & 2 \le |x| < 3 \\ 0 & 3 \le |x| \end{cases} \tag{21.7}$$

(L=6, 8, 16)

$$i(x) = sinc(x) rect(\frac{x}{L}) \tag{21.8}$$

Chapter 22

FILTRANGE

In this chapter the processing of step FILTRANGE is described. This *optional* step filters the spectra in range direction of master and slave to reduce noise in the interferogram. The noise reduction results from filtering out non overlapping parts of the spectrum. This spectral non overlap in range between master and slave is caused by a slightly different viewing angle of both sensors. The longer the perpendicular baseline, the smaller the overlapping part. Eventually a baseline of about 1100 m results in no overlap at all (the critical baseline for ERS). (Assuming no local terrain slope.) A reduction of typically 10-20% in the number of residues can be achieved.

Method porbits filters based on the orbits (perpendicular baseline) for a constant (given) terrain slope. Perform this step after coarse coregistration, since the approximate overlap is used to filter both images. The output images are cropped to this overlap. To filter 'on the save side', i.e., not to filter out too much, use a negative terrain slope of, e.g., 10 degrees.
This step is not recommended, except perhaps to improve the coregistration polynomial for long baseline pairs. After the resampling the range filtering then could be repeated on the original data with the adaptive algorithm.

Method adaptive should be performed after the resampling of the slave to the master grid, because the fringe frequency is estimated from the interferogram (that is temporary computed). It is performed simulataneous for the master and slave image.

22.1 Input Cards

RF_METHOD *adaptive* | *porbits*
Method selector for range filtering. Either adaptive (recommended) or based on the precise orbits.

RF_FFTLENGTH *64*
For method porbits and adaptive. For method porbits: length of block in range direction, 512 or 1024 (default for this method) advised. For method adaptive: Length of window for adaptive method. A peak is estimated for parts of this length.

RF_OVERLAP *0*
For method adaptive. Overlap between input buffers in range direction.

RF_HAMMING *0.75*
For method porbits and adaptive. Weight for hamming filter (1 is rect).

RF_SLOPE *0*
For method porbits. Terrain slope in degrees. Positive slope is towards radar. A slope equal to the viewing angle implies total filtering.

RF_NLMEAN *15*
For method adaptive. Take (walking) mean over RF_NLMEAN lines to reduce noise for peak estimation. Has to be odd. Compare with periodogram.

RF_THRESHOLD *5*
For method adaptive. Threshold on SNR of peak estimation to perform range filtering.

RF_OVERSAMPLE *2*
For method adaptive. Oversample master and slave with this factor before computing the complex interferogram for peak estimation. This factor has to be a power of 2. 2 is default to be able to estimate the peak for frequency shifts larger than half the bandwidth. A factor of 4 for example might give a better estimate, since the interval between shifts that can be estimated is in that case halfed (fixed FFTLENGTH).

RF_WEIGHTCORR [*ON* | *OFF*]
For method adaptive. In peak estimation, weight values to bias higher frequencies. The reason for this card is that the low frequencies are (for small OVERSAMPLE factors) aliased after interferogram generation. The deweighting is done by a dividing by a triangle function (convolution of 2 rect functions, the shape of the range spectrum). Effect of this card may be neglectable.

RF_OUT_MASTER *master.rfilter*
Output data file name of master.

RF_OUT_SLAVE *slave.rfilter*
Output data file name of slave.

RF_OUT_FORMAT *cr4* | *ci2*
Output data format for master and slave file.

Example input:

```
 c
 c
comment   ___ ADAPTIVE RANGE FILTERING ___
 c
RF_METHOD       adaptive
RF_FFTLENGTH    128                              // 2500 m
RF_NLMEAN       15                               // odd
RF_THRESHOLD    5                                // SNR
RF_HAMMING      0.75                             // alpha
 RF_OVERSAMPLE   4
RF_WEIGHTCORR   OFF
RF_OUT_MASTER   Outdata/3397.rfilter
RF_OUT_SLAVE    Outdata/23070.rfilter
RF_OUT_FORMAT   ci2
```

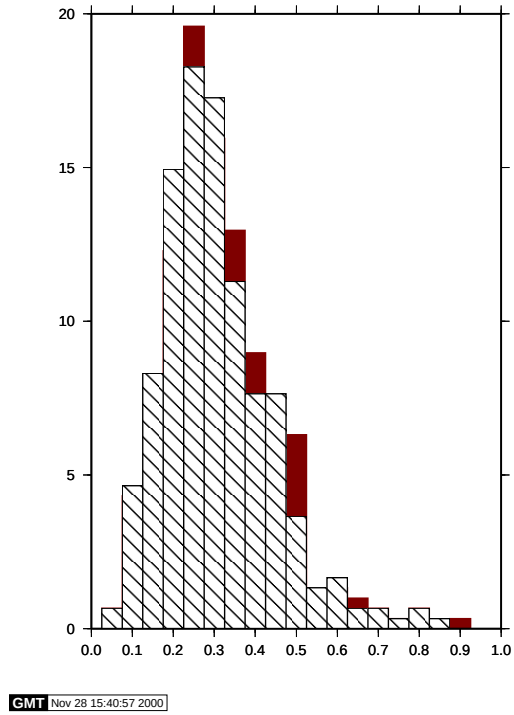

Figure 22.1: Frequency histograms of the FINE coregistration correlation values. At 301 locations of a image the fine coregistration was performed, and a histogram has been made of the correlation values. The bin width equals 0.05. It can be clearly seen that the histogram of the filtered data is located to the right, i.e., is better.

22.2 Output Description

The **process control flags** in the **result files** for the master and the slave are switched on:

```
filt_range:             1
```

In the filtrange section the filename and format is described after the filtering. Master and slave are cut out so they exactly fit on each other.

```
*****************************************************************
*_Start_filt_range:
*****************************************************************
Method:                                 adaptive
Data_output_file:                       Outdata/23070.rfilter.3
Data_output_format:                     complex_real4
First_line (w.r.t. original_master):    201
Last_line (w.r.t. original_master):     6000
First_pixel (w.r.t. original_master):   100
Last_pixel (w.r.t. original_master):    3000
*****************************************************************
* End_filt_range:_NORMAL
*****************************************************************
```

Figure 22.1 shows the improvement in the correlation for method porbits. A histogram is made of the correlation values of the FINE coregistration with and without range filtering. The area was relatively flat and the perpendicular baseline was approximately 175 meters.

22.3 Implementation

22.3.1 Method: porbits

The frequency shift Δf between the master and slave range data spectra equals

$$\Delta f = -\frac{c}{\lambda}\frac{B_\perp}{r_1 \tan(\theta-\alpha)} = -\frac{c}{\lambda}\frac{\tan(\Delta\theta)}{\tan(\theta-\alpha)} \approx -\frac{c}{\lambda}\frac{\Delta\theta}{\tan(\theta-\alpha)} \tag{22.1}$$

Where:
$\Delta\theta = \theta_1 - \theta_2$, α is the local terrain slope w.r.t. the ellipsoid, c is the speed of light, θ is the local incidence angle (!), λ is the radar wavelength, r_1 is the slant range ground to master. The approximation is used in Doris. Of course, the sign of $B_\perp$, or $\Delta\theta$ is important to filter the correct side of the spectra. Note that

$$\alpha \to 23^\circ \Leftrightarrow \Delta f \to \infty \tag{22.2}$$

The local incidence angle is computed with the dot product of vectors P, and P-M. See also [Gatelli et al., 1994].

The algorithm in Doris works as

- While there is a line in the overlap, get next line for master and slave.
- Get block of FFT_LENGTH pixels.
- Compute viewing angle, perpendicular baseline, delta theta for middle pixel of block.
- Compute frequency shift by equation 22.1, and compose filter of rect and hamming.
- Filter master and slave.
- Write block back (for last block only partially).

22.3.2 Method: adaptive

After the resampling of the slave on the master grid is performed this algorithm can be used. The local fringe frequency is estimated using peak analysis of the power of the spectrum of the complex interferogram. The resampling is required since the local fringe frequency is estimated from the interferogram. This fringe frequency is directly related to the spectral shift in range direction. (Note this shift is not a shift, but different frequencies are mapped on places with this shift...) The algorithm generally works as follows.

- Take part of master and slave (e.g., 500 lines by 128 range pixels.)
- Oversample master and slave and generate complex interferogram.
- Take FFT over range for all lines of complex interferogram.
- Take power. If requested, weight this powerspectrum with auto-convolution of 2 rect functions with appropriate bandwidth. (Actually, perhaps the spectrum should also be weighted with autoconvolution of Hamming, but since I am not sure that this has a big impact on real data this is not done.)
- Take moving average over the lines of the power FFT's for noise suppression (kind of periodogram). (This was better implemented as a convolution with a block function (e.g., 9 x 128)?)
- Estimate peak per line in oversampled/averaged powerspectrum of complex interferogram. Estimate $SNR = \frac{\mathrm{fftlength}\cdot peak}{rest}$.

- This peak is directly related to overlap of spectra for this part of this line. (See also Fig. 22.2.) $\Delta f_r = f_{\text{fringe}}$.
- If SNR is above threshold (input of user, e.g., 3), remove appropriate parts of spectra of master/slave. Optionally compute inverse hamming window and new hamming window, and rect window to filter one side of master spectrum, and other side of slave spectrum. (See also Fig. 22.3.) Note that the filter is mirrored (matlab fliplr) for master/slave. The SNR of the peak of a random spectrum (sea) probably is a little larger than 1, so threshold of 3 may not be large enough.
- Do inverse FFT for filtered master, slave, which yields the filtered image in the space domain.
- Take next part of master and slave (e.g., 500 lines by next 128 range pixels) until all lines are filtered.

In practice this is done in blocks. These blocks are overlapping in lines (because the averaging over lines means one cannot filter all lines in the block), and not in range. Parameters that can be adjusted are the FFTlength, the moving average mean, the SNR threshold.

The fftlength should be large enough to yield a good estimate of the local fringe frequency, and small enough to contain a constant slope of the terrain. The total number of fringes in range directorion can be easily estimated using the perpendicular baseline.

It is probably a good idea to add a card so an overlap in range between blocks can be used. This avoids 'edge' effects, and increases the filtering of terrain near, e.g., a lake (since the SNR for peak detection will be higher for a number of blocks towards the noise). This is not implemented yet.

See also [Gatelli et al., 1994], [Geudtner, 1996], [Curlander and McDonough, 1991]. See also our matlab toolbox.

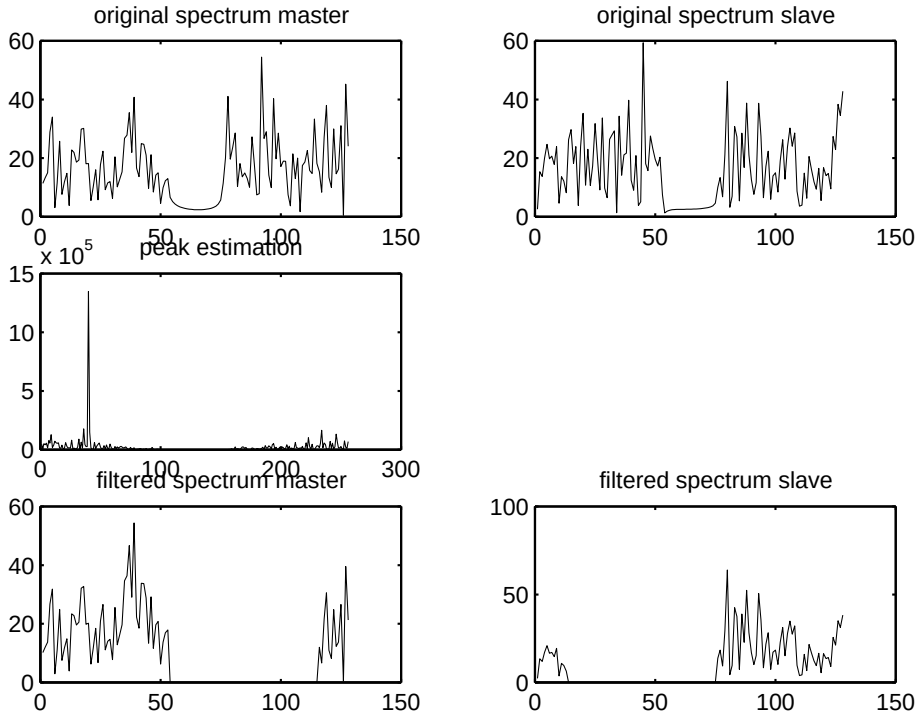

Figure 22.2: Peak estimation in spectral domain of (oversampled) complex interferogram. Non FFTshifted.

22.3.3 Hamming filter

The Hamming filter that optionally is used to de-weight en re-weight the spectrum of master and slave has the form:

$$W(f_r) = \left[\alpha + (1-\alpha)\cos(2\pi\frac{f_r}{f_s})\right] \mathrm{rect}\left(\frac{\mathrm{f_r}}{\mathrm{B_r}}\right). \tag{22.3}$$

Where f_r is the frequency axis (-fs/2:df:fs-df, df=fs/N). f_s is the range sampling rate (18.96MHz), and B_r is the bandwidth in range (15.55MHz). α is a parameter controlling the amount of weighting.

$$\mathrm{rect(x)} = \begin{cases} 1, & \| x \| < 0.5 \\ 0, & \mathrm{otherwise} \end{cases} \tag{22.4}$$

Note: rect not periodic.

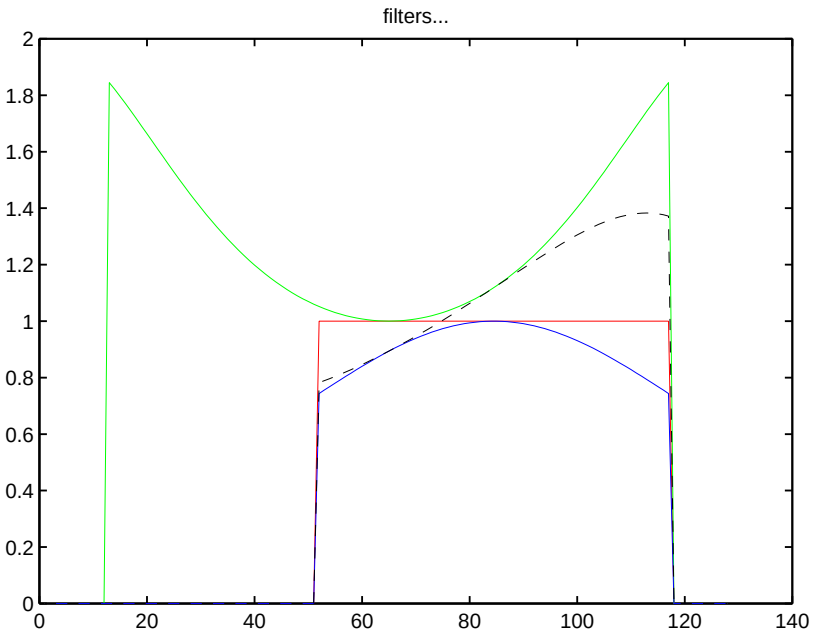

Figure 22.3: Spectral filtering windows (inverse hamming, boxcar (rect), and new hamming. Note these are FFTshifted.

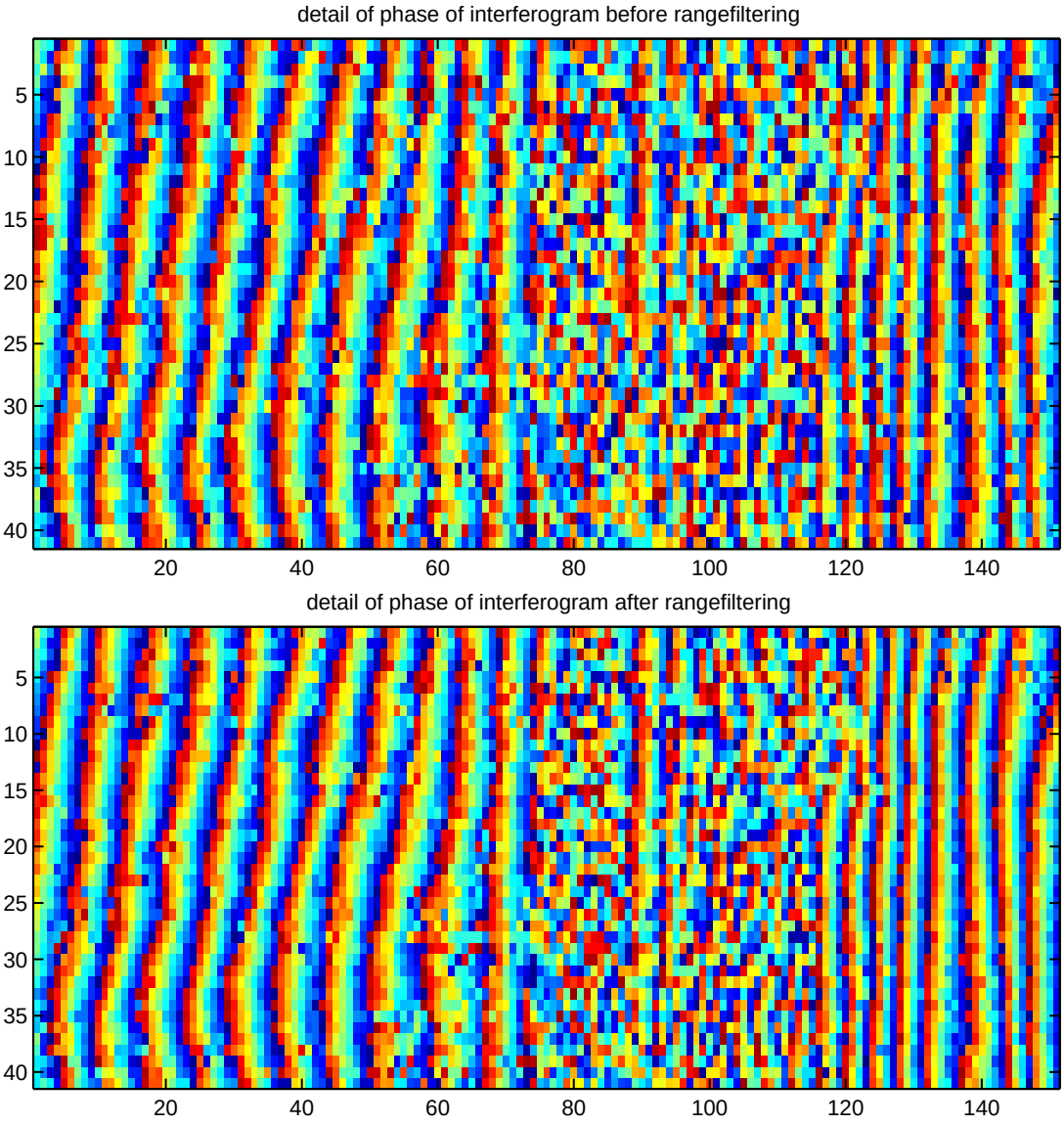

Figure 22.4: Detail of interferogram with and without rangefiltering. (fftlength=128, nlmean=15, snrtreshold=5). The perpendicular baseline is about 200 m for this interferogram. The fringes are clearly sharper after the filtering. The number of residues for the interferogram was reduced by 20%. Subtraction of both interferograms yielded a random phase, so no structural effect of range filter implemenation is suspected.

Chapter 23

INTERFERO

In this chapter the processing of step INTERFERO is described. In this step the following is computed.

The (complex) interferogram is computed, with or without subtraction of the reference phase. The reference phase is subtracted if there is a 2d-polynomial in the **products result file** (result of step FLATEARTH). It is not subtracted if this is not in the **result file** or if the number of coefficients is set to 0.

The complex interferogram minus reference phase is defined as:

$$I = M \cdot S^* \cdot R^* \tag{23.1}$$

Where:
$\{.\}^*$ denotes the complex conjugated;
$\cdot$ denotes a pointwise multiplication;
I is the complex interferogram;
M is the complex master image;
S is the complex (resampled) slave image;
R is the complex (amplitude $\equiv$ 1) reference phase.

The phase image (of the complex interferogram minus reference phase) is defined as:

$$\phi = \arctan_2(I_{\text{imag}}, I_{\text{real}}) \tag{23.2}$$

Where:
$\arctan_2$ is the four quadrant arc tangent;
ϕ is the phase image;
I is the complex interferogram;

This is identical to

$$\phi = \phi_M - \phi_S - \phi_R \tag{23.3}$$

Multilooking can be performed to reduce noise. Usually a ratio of (line:pixel) = 5:1 between the factors is chosen to obtain more or less square pixels (20x20m^2 for factors 5 and 1). (The resolution decreases of course if multilooking is applied.)

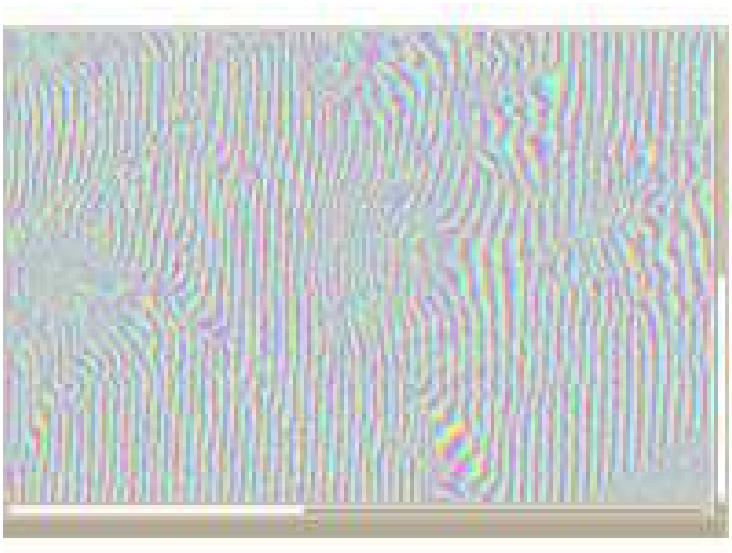

Figure 23.1: Phase image of complex interferogram. The 'flat earth' is clearly visible as a trend over the interferogram.

23.0.4 NEW

Note that for the optimal results, i.e. to avoid an aliasing of spectras, the images first have to be oversampled by a factor two before multiplication for an optimal result. See sections on 'OVERSAMPLE' card.

For a more modular approach in the new method it is not advisable to subtract the reference phase in this step. If you do want to subtract the reference phase here, then make sure you first run Doris to run comprefpha, and then make a second run for step interfero.

After generation of the complex interferogram, the reference phase can be computed by the new module comprefpha and subtracted by the new module subtrrefpha. (Also a reference height model can be computed and subtracted in future modules.)

Figure 23.1 shows an example of a complex interferogram. Only the phase is shown here. We processed orbits 21066 and 1393 of frame 2781 (Italy), acquired at 26th and 27th July 1995 respectively (ERS1,2 Tandem mission). The parallel baseline is about 35 meters, which implies a height ambiguity of about 270 meters. Clearly a large trend caused by the 'flat earth' is present, but also some topographic features can be seen. In the frame the elavation ranges from zero to 1400 meters. (The original SLC images were cut out to 20000 lines by 4000 pixels, The interferogram is multiooked by factors 10 in azimuth and 2 in range, which yields a dimension of 1475 lines by 1997 pixels.)

23.1 Input Cards

INT_OUT_CINT *filename*
filename of output datafile for complex interferogram (of step interferogram). one of INT_OUT_* is mandatory.

INT_OUT_INT *filename*
filename of output datafile for (real) interferogram (of step interferogram). one of INT_OUT_* is mandatory.

INT_MULTILOOK *5 1*

multilookfactor, if no multilooking is desired, set this to ”1 1”. If the reference phase is not subtracted in this step, be carefull not to multilook too much in this step. In step subtrrefpha again a multilook card is present (where one can multilook by factors 2 2 for example).

Example input section:

```
 c
 c
 comment ___product generation___
 c
INT_OUT_CINT    Output/cint.raw                // optional
 c INT_OUT_INT   Output/int.raw                 // optional
 c INT_OUT_FE    Output/flatearth.raw           // optional
INT_MULTILOOK   10 2                           // line, pixel
```

23.2 Output Description

At successful exit, the **process control flag** is switched on:

```
interfero:              1
```

The output looks like (in the **products result file**):

```
*****************************************************************
*_Start_interfero
*****************************************************************
Data_output_file:                       Output/cint.raw
Data_output_format:                     complex_real4
First_line (w.r.t. original_master):    1001
Last_line (w.r.t. original_master):     2105
First_pixel (w.r.t. original_master):   501
Last_pixel (w.r.t. original_master):    700
Multilookfactor_azimuth_direction:      10
Multilookfactor_range_direction:        2
Number of lines (multilooked):          110
Number of pixels (multilooked):         100
*****************************************************************
* End_interfero:_NORMAL
*****************************************************************
```

The (complex) output data file can be viewed with, e.g., in Matlab with a following script:

```
fid = fopen('Output/cint.raw','r');
cint = (fread(fid,[100 220],'float32')).';
fclose(fid);
realpart = cint[:,1:2:size(cint,1)];
cplxpart = cint[:,2:2:size(cint,1)];
cint = realpart + i*cplxpart;
phase = angle(cint);
imagesc(phase);
colorbar;
```

Output may include the matrix with the reference phase. For flatearth correction this normally resembles a plane, and is not very useful output.

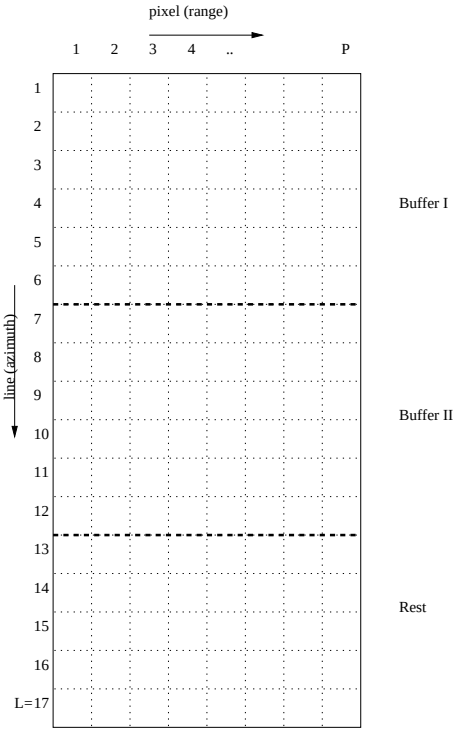

Figure 23.2: Use of buffers in implementation of computation interferogram.

23.3 Implementation

The following is computed (in buffers).

1. Read in (buffer of) complex master and slave image (M and S).
2. If there is a reference phase R (real values), evaluate it at the master grid (buffer), and then subtract it (in a complex way) from slave S and store it (in S). (matlab like notation, pointwise multiplication notated by .*)

$$S = S. * (\cos R + i \sin R) \tag{23.4}$$

3. Compute complex interferogram and store it in M.

$$M = M. * conj(S) (\equiv M. * conj(S). * conj(R)) \tag{23.5}$$

4. Multilook complex interferogram if requested. Do not divide (scale) multilooked interferogram.
5. Write this buffer to disk and start next one.

See Figure 23.2 for an explanation of the use of buffers in the implementation. In this example, the number of lines equals 17. The number of pixels equals P. Suppose multilook factor in line direction mL=3 and in pixel direction mP=3. Furthermore, after computing the available memory, the number of lines of one buffer (BL) is maximum 7. BL is set to a multiple of mL, BL=6.

Now the number of fully filled buffers NB = 17/6 = 2. The number of lines left in the last buffer equals 17We can compute something in this last buffer only for the first three lines, so the last buffer is 3 lines long.

The first buffer is read (master and slave image), and computations are performed. These results are written to disk. Then next buffer, etc. If buffers==3 then resize the matrices for the computations.

The number of lines of the total result are L/mL and P/mP (floored).

Chapter 24

COMPREFPHA

In this chapter the processing of step COMPREFPHA is described. This step can be performed as soon as the precise orbits are known. The recommended approach is to compute this only after the computation of the interferogram, and then use the step SUBTRREFPHA to subtract it. This step is not required if method ”exact” is used in step SUBTRREFPHA.

The flatearth correction (the phase caused by the reference surface (WGS84 for now)) is computed in this step. For a certain (line,pixel) in the master image the corresponding coordinates (x,y,z) of the master and slave satellite and the point P on the reference ellipsoid are computed, utilizing the set of equations (Doppler, range and ellipsoid equation), see annex D. Then the parallel baseline and the phase is computed.

The parallel baseline $B_{\|}$ is defined as (M/S are positions of master/slave, P is the position of the point on the reference surface):

$$B_{\|} = d(M,P) - d(S,P) \tag{24.1}$$

The phase of a pixel in the master image is defined as:

$$\phi = -\frac{4\pi}{\lambda} d(M,P) \tag{24.2}$$

The reference phase for this pixel is defined as:

$$\phi = -\frac{4\pi}{\lambda} B_{\|} \tag{24.3}$$

The reference phase is computed in a number of points distributed over the total image in this way, after which a 2d-polynomial is estimated (least squares) fitting these ’observations’. (So a plane can be fitted by setting the degree to 1.)

A 2d-polynomial is defined as:

$$f(x,y) = \sum_{i=0}^{d} \sum_{j=0}^{i} \alpha_{i-j,j} x^{i-j} y^{j} \tag{24.4}$$

Thus the order of the coefficients (line,pixel) is (degree d):
d=0: A_{00} (1)
d=1: $A_{10} A_{01}$ (2, 3)
d=2: $A_{20} A_{11} A_{02}$ (4, 5, 6)
d=3: $A_{30} A_{21} A_{12} A_{03}$ (7, 8, 9, 10)

Thus the number of coefficients (unknowns) equals:

$$\frac{1}{2}((d+1)^2 + d + 1) \tag{24.5}$$

A polynomial of degree 5 normally is sufficient to model the reference phase for a full scene. A lower degree might be selected for smaller images, which also should increase the stability of the normalmatrix. We would expect the higher order terms to be small because the polynomial describes a smooth, long wave body (ellipsoid). To force the polynomial to be smooth one might consider always using a polynomial of degree 2 or 3.

See also [Schwäbisch, 1995] or [Geudtner and Schwäbisch, 1996].

24.1 Input Cards

FE_METHOD *porbits*
Method selector for this step. currently there is only one method.

FE_DEGREE *5*
degree of 2d polynomial.

FE_NPOINTS *501*
Number of points to compute reference phase for least squares estimation.

FE_IN_POS *filename*
ascii file with positions (master coord. system) in it where to compute the reference phase, and then to model it by a polynomial. Card FE_NPOINTS is ignored if this card is specified. This card can be used, e.g., if it is desired to have the points on a grid, including the edges. Possibly one might even select points outside the grid (though not smaller than 0), in order to avoid excessive fluctuations at the edge if a higher degree polynomial is used.

FE_OUT_FILE *filename*
Card will be added in future for optional output of reference phase.

One can use an awk like to make a grid:

```
awk 'BEGIN{for (i=100;i<25200;i=i+500)   \
            {for (j=750;j<5400;j=j+200)  \
              {printf "%i %i \n",i,j}}   \
     exit}' > positions.in
```

and a card in the **input file**:

```
FE_IN_POS   positions.in
```

This step used to be named "FlatEarth". This explains the prepended FE_'s, instead of some abbreviation for reference phase.

Example input section:

```
 c
 c
 comment ___ COMPREFPHA ___
 c
FE_METHOD       porbits
FE_DEGREE       3
FE_NPOINTS      201
```

24.2 Output Description

At successful exit the **process control flag** is switched on:

```
comp_refpha:             1
```

The output (in the **products result file**) looks like:

```
*********************************************************************
*_Start_comprefpha
*********************************************************************
Degree_flat:    5
Estimated_coefficients_flatearth:
 5.17144173e+03         0 0
 4.03705656e-03         1 0
 2.17736976e-01         0 1
-2.05452064e-06         2 0
 2.15880157e-07         1 1
-1.27934869e-05         0 2
 8.80499980e-10         3 0
-1.64339133e-11         2 1
-8.28354289e-11         1 2
-6.04376860e-10         0 3
-1.64239900e-13         4 0
 2.25037286e-15         3 1
 1.48243991e-14         2 2
 5.52622526e-14         1 3
 1.74773307e-12         0 4
 1.11724810e-17         5 0
 2.74597475e-20         4 1
-2.64743817e-18         3 2
 4.10973605e-18         2 3
-3.09581184e-17         1 4
-6.94891272e-16         0 5
*********************************************************************
* End_comprefpha:_NORMAL
*********************************************************************
```

In the log file, some statistics are given for the errors (observation minus estimated value). These errors should have a maximum of 0.1 phase cycle. The can be plotted with GMT or someother package to evaluate the difference between the polynomial and the 'observations'.

Here also the standard deviation per estimated coefficients is given. This std. seems to be too large, but an error in the computations could not be found.

The polynomial can not easily be visualized at the moment. (It is normalized, not evaluated in this step.) In the old method of the interfero step (see Chapter 23), there was a card to output the reference phase polynomial because it was evaluated there anyway. It seems logical to add a card for outputting the (wrapped or unwrapped) reference phase in the step subtrrefpha (Chapter 25), which likely will be added in one of the cumming releases.

Chapter 25

SUBTRREFPHA

In this chapter the processing of step SUBTRREFPHA is described.

This step requires the steps INTERFERO and COMPREFPHA for obvious reasons. In this step the reference phase of a mathematical body (ellipsoid) is subtracted from the complex interferogram. This is done by complex multiplication with the conjugated, written symbolically as follows:

$$I = I \cdot (\cos R_\phi - i \sin R_\phi) \tag{25.1}$$

Where I is the complex interferogram, $\cdot$ denotes pointwize multiplication, and $R\phi$ is the reference phase for a certain point.

25.1 Input Cards

SRP_METHOD *polynomial* | *exact*
method selector for subtraction of reference phase. ”polynomial” evaluates the polynomial computed in step COMP_REFPHA, ”exact” computes the reference phase explicitly for each pixel, and subtracts it. Computations are done by evaluation system of 3 equations foreach pixel.

SRP_OUT_CINT *cint.minrefpha.raw*
filename of output datafile for complex interferogram (of step subtrrefpha).

SRP_MULTILOOK *1 1*
multilook factors in line (azimuth) and pixel (range) direction.

SRP_DUMPREFPHA [*OFF* | *ON*]
This card specifies to dump the reference phase as a complex real4 file, containing the evaluated reference phase polynomial just as it would have been subtracted from the complex interferogram (multilooked). The amplitude should be equal to one by definition. WARNING: the reference phase is not subtracted, only dumped, if this card is specified. If you want to study the reference phase for different ellipsoids, compile different versions of Doris, changing the parameters in the file refsystem, and use these executables to generate the reference phase.

SRP_OUT_REFPHA *refphase.raw*
name of **output file** reference phase. Only if SRP_DUMPREFPHA.

SRP_OUT_H2PH *filename*

Request optional output of height-to-phase factors.

Example input section for dumping the reference phase:

```
 c
 c ___ step subtrrefpha ___
 c SRP_METHOD      exact
 c
SRP_METHOD      polynomial
 c SRP_OUT_CINT    Outdata/cint.minrefpha.raw
SRP_MULTILOOK   4 4
SRP_DUMPREFPHA
SRP_OUT_REFPHA  Outdata/refpha.raw
```

25.2 Output Description

At successful exit the **process control flag** is switched on:

```
subtrrefpha:             1
```

The output (in the **products result file**) looks like:

```
*****************************************************************
*_Start_subtrrefpha:
*****************************************************************
Data_output_file:                       Outdata/cint.minrefpha.raw
Data_output_format:               complex_real4
First_line (w.r.t. original_master):    245
Last_line (w.r.t. original_master):     14964
First_pixel (w.r.t. original_master):   7
Last_pixel (w.r.t. original_master):    3998
Multilookfactor_azimuth_direction:      40
Multilookfactor_range_direction:        8
Number of lines (multilooked):          368
Number of pixels (multilooked):         499
*****************************************************************
* End_subtr_refphase:_NORMAL
*****************************************************************
```

We noticed that if the precise orbits are not long enough (not enough time before and after first/last line), this results in a wrong reference phase for obvious reasons. Interpolation near the end of the data points is not very good with cubic splines. This can be solved by using more orbital data points after the last line of the scene (see cards for step M_PORBITS).

Figure 25.1 shows the result of subtracting the reference phase polynomial from the interferogram (Figure 23.1). (The same scene as described in section 23.0.4, again multilooked, now by factors 4 and 4, resulting in total multilooking of 40 and 8, which agrees on the terrain with a resolution of about 160 meters square.)

This step can be mis-used to correct for residual orbital fringes (if you know what you are doing).

To do this, first count the number of fringes you want to remove from the interferogram. Then, edit the **products result file** and create a section for the step COMPREFPHA. In this output section, simply define a polynomial that describes for example a linear trend in range of, say, 2.5 fringes. Then, run this step, and doris will not know that it is not the reference phase polynomial that is subtracted from the interferogram, but an additional correction polynomial that has been inserted by hand.

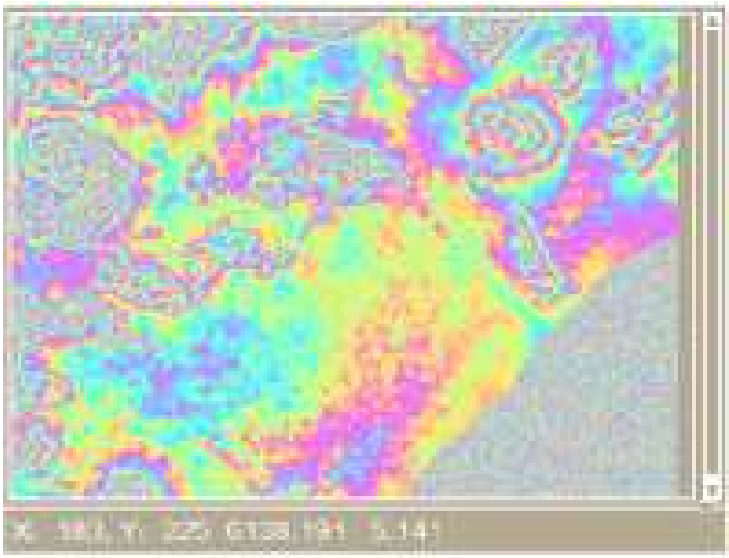

Figure 25.1: Phase image of complex interferogram. The 'flat earth' is subtracted, leaving dominantly topographic fringes.

Chapter 26

COMPREFDEM

In this chapter the processing of step COMPREFDEM is described. A DEM is radarcoded at the grid of the interferogram. In step subtrrefdem (chapter 27) it can be subtracted from the complex interferogram.

This step requires a DEM. It can best be performed after the interferogram generation. SRTM can be used to obtain the topography in 3 arcseconds (Near global: $\sim$ 90 m. at the equator) or 1 arcsecond (USA only) resolution. The input DEM file must have the byte order of your platform in order to extract correct elevation value, see CRD_IN_FORMAT option for details. The DEM should be in the WGS84 system (same as the orbit ephemerides) The Doris distribution contains the utility *construct_dem.sh* to download and prepare SRTM data (see Section C.2.12).

An alternative is the USGS gtopo30 DEM. This is a global DEM with relatively low precision and gridspacing (30 seconds, approximately 1 km at equator). There are 33 tiles covering the globe, in total requiring about 3GB of disk space. (For more information, see for example `http://edc.usgs.gov/products/elevation/gtopo30/gtopo30.html` or Google for more information).

The input DEM has to be in an equiangular grid. The format is short signed integers, real4 floats, or real8 doubles (meters). The DEM should be in WGS84 system (same as orbit ephemerides). The UpperLeft pixel of the DEM matrix on file should be the most North, most West pixel, i.e., the pixel with largest latitude (between -90,90) and smallest longitude (between -180,180 (?)).

26.1 Input Cards

CRD_IN_DEM *filename*
filename of input DEM. File is assumed to be stored in a raster. Major row order. from North to South, line-by-line. See also internet links at Doris home page for available DEMs.

CRD_IN_FORMAT *I2* | *I2_BIGENDIAN* | *R4* | *R8*
format of input DEM on file (signed short for gtopo30, or real4, or real8; input matrix is raw binary data w/o header, endianness of host platform is assumed, except for I2_BIGENDIAN).

CRD_IN_SIZE *6000 4800*
Number of rows and columns of input DEM file. Default is set to tile w020n90.DEM.

CRD_IN_DELTA *0.00833333333333333333* [*deltalon*]

Grid spacing of input DEM in decimal degrees, latitude longitude. Default is equal gridspacing, default set to tile w020n90.DEM.

CRD_IN_UL *89.995833333333333 -19.995833333333333333333*
Coordinates of UL (upperleft) corner in decimal degrees, latitude [-90, 90] longitude [-180, 180]. Default is set to tile w020n90.DEM. It is interpreted as max-latitude, min-longitude in source.

CRD_IN_NODATA *-9999*
Identifier to ignore data in input DEM with this value. Default is set to tile w020n90.DEM.

CRD_INCLUDE_FE *OFF* | *ON*
If this card is switched on, the reference phase of the DEM is computed, including the 'flat earth' term. Otherwize, the phase is computed with respect to the ellipsoid, yielding only topographic phase.

CRD_OUT_DEM *filename*
Request optional debug output to float file of input DEM per buffer, cut to the interferogram window. Info on these files is written as DEBUG.

CRD_OUT_DEMI *filename*
Request optional debug output to float file of interpolated input DEM, cut to the interferogram window. Info on these files is written as DEBUG.

CRD_OUT_FILE *refdem.raw*
Filename of output radarcoded DEM.

CRD_OUT_DEM_LP *filename*
Request optional output of DEM in radar coordinates.

CRD_OUT_H2PH *filename*
Request optional output of height-to-phase factors.

Most of these parameters can be found in the .HDR file of gtopo30 DEM's. Example input section:

```
#   ----------------------------------------
#      REFERENCE DEM
#   ----------------------------------------

CRD_IN_DEM   final_wanaF2835.dem
CRD_IN_FORMAT r4         // default is short integer
CRD_IN_SIZE  3601 3601
CRD_IN_DELTA  0.000833333  0.000833333
CRD_IN_UL    40  27
CRD_IN_NODATA  -32768
CRD_OUT_DEM_LP    42408_22735.demlp
CRD_OUT_FILE   42408_22735.demphase
CRD_OUT_H2PH   42408_22735.h2ph
```

26.2 Output Description

At successful exit the **process control flag** is switched on:

```
comp_refdem:            1
```

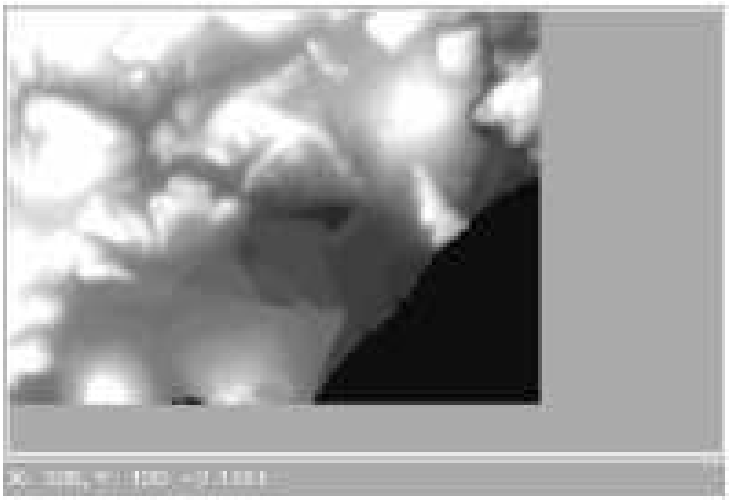

Figure 26.1: Interferogram of radarcoded DEM for area described in section 23.0.4.

The output (in the **products result file**) looks like:

```
*****************************************************************
*_Start_comp_refdem:
*****************************************************************
Include_flatearth:                      No
DEM source file:                        /d2/doris-test/final_wanaF2835.dem
Min. of input DEM:                      92
Max. of input DEM:                      1791
Data_output_file:                       42408_22735.demphase
Data_output_format:                     real4
First_line (w.r.t. original_master):    3060
Last_line (w.r.t. original_master):     8052
First_pixel (w.r.t. original_master):   1719
Last_pixel (w.r.t. original_master):    2710
Multilookfactor_azimuth_direction:      1
Multilookfactor_range_direction:        1
Number of lines (multilooked):          4993
Number of pixels (multilooked):         992
*****************************************************************
* End_comp_refdem:_NORMAL
*****************************************************************
```

The output in the logfile is more verbose, specifying the results of the intermediate steps. Also go over the standard out in case of problems, with the SCREEN set to DEBUG level.

Figure 26.1 shows an example of a real valued phase map (interferogram) for a radarcoded DEM (the Data_output_file). (This applies to the same scene as described in Section 23.0.4.)

26.3 Implementation

The DEM reference phase is computed in two steps:

First, the DEM is radarcoded to the coordinate systems of the master image. Per DEM point the master coordinate (real valued) and the computed reference phase is saved to a file.

Second, the reference phase is interpolated to the integer grid of master coordinates. A linear interpolation

based on a Delaunay triangulation is used. The software package *Triangle* for the Delaunay triangulation is kindly made available by Jonathan Shewchuk [Shewchuk, 1996, Shewchuk, 2002], see also `http://www.cs.cmu.edu/~quake/triangle.html`.

Chapter 27

SUBTRREFDEM

In this chapter the processing of step SUBTRREFDEM is described.

This step requires the steps INTERFERO and COMPREFDEM for obvious reasons. In this step the reference phase of a digital elevation model is subtracted from the complex interferogram. This is done by complex multiplication with the conjugated, as explained in step subtrrefpha (chapter 25).

27.1 Input Cards

SRD_OUT_CINT *cint.minrefdem.raw*
Filename of output complex interferogram.

SRD_OFFSET *0 0*
Offset to be applied in azimuth line, range pixel direction. The synthetic phase image outputed by COMPREFDEM is shifted by the specified offset before subtraction. A positive shift indicates a shift of the synthetic phase image to the right (range), up (azimuth).

Example input section:

```
c
c ___ step subtrrefdem ___
SRD_OUT_CINT    Outdata/cint.minrefdem.raw
SRD_OFFSET      1 -2
```

27.2 Output Description

At successful exit the **process control flag** is switched on:

```
subtr_refdem:            1
```

The output (in the **products result file**) looks like:

```
*****************************************************************
*_Start_subtrrefdem
*****************************************************************
Additional_azimuth_shift:              1
Additional_range_shift:                -2
```

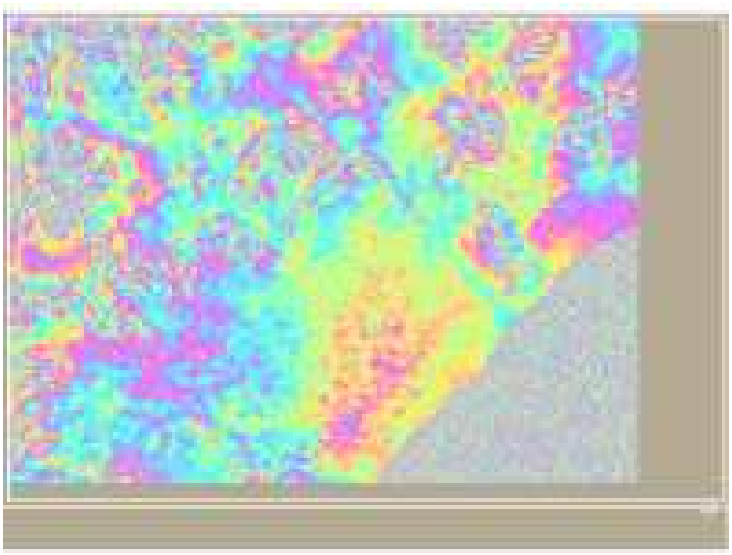

Figure 27.1: Phase image of complex interferogram. The phase of the reference DEM is subtracted ('flat earth' was already subtracted in subtrrefpha), leaving residual topographic, atmospheric, and errorous fringes.

```
Data_output_file:                       Outdata/cint.minrefdem.raw
Data_output_format:                     complex_real4
First_line (w.r.t. original_master):    245
Last_line (w.r.t. original_master):     14964
First_pixel (w.r.t. original_master):   7
Last_pixel (w.r.t. original_master):    3998
Multilookfactor_azimuth_direction:      40
Multilookfactor_range_direction:        8
Number of lines (multilooked):          368
Number of pixels (multilooked):         499
***************************************************************
* End_subtrrefpha:_NORMAL
***************************************************************
```

Figure 27.1 shows the result of subtracting the reference phase of the radarcoded DEM from the interferogram (See Figure 25.1). It can be seen the number of topographic fringes is reduced, though there still remain some residual effects. A next version of Doris will include an option to first correlate the radarcoded DEM with the interferogram, to find an additional offset. The radarcoded reference DEM is then shifted on (multilooked) pixel level before subtraction.

Chapter 28

COHERENCE

In this chapter the processing of step COHERENCE is described.

In this step the following is computed. The (complex) coherence image is computed, with or without subtraction of the reference phase and the radarcoded DEM phase. The reference phase is subtracted if there is a 2d-polynomial in the **products result file** (result of step FLATEARTH). It is not subtracted if this is not in the **result file** or if the number of coefficients is set to 0. For a proper estimation of the coherence map, especially over mountainous areas, the radarcoded DEM phase should also be subtracted. In this case, the result of step COMPREFDEM is required in the products result file. This is the default procedure since v4.01.

The complex coherence image between two images is defined as:

$$\gamma_c = \frac{E\{M \cdot S^*\}}{\sqrt{E\{M \cdot M^*\} \cdot E\{S \cdot S^*\}}} \tag{28.1}$$

Where:
$E\{.\}$ is the expectation;
* is the complex conjugated;
γ_c is the complex coherence;
M is the complex master image;
S is the complex slave image (possibly minus (complex) reference phase and DEM phase: $S = S \cdot R^*$).

The coherence is defined by $|\gamma_c|$ and its estimator as:

$$\hat{\gamma} = \left| \frac{\frac{1}{N}\sum_{i=0}^{N} M_i S_i^*}{\sqrt{\frac{1}{N}\sum_{i=0}^{N} M_i M_i^* \frac{1}{N}\sum_{i=0}^{N} S_i S_i^*}} \right| \tag{28.2}$$

Multilooking can be performed to reduce noise. Usually a ratio of (line:pixel) = 5:1 between the factors is chosen to obtain more or less square pixels (20x20m^2 for factors 5 and 1). (The resolution decreases of course if multilooking is applied.)

Data buffering is applied for the memory considerations.

28.1 Input Cards

COH_METHOD	*include_refdem*	*refphase_only*

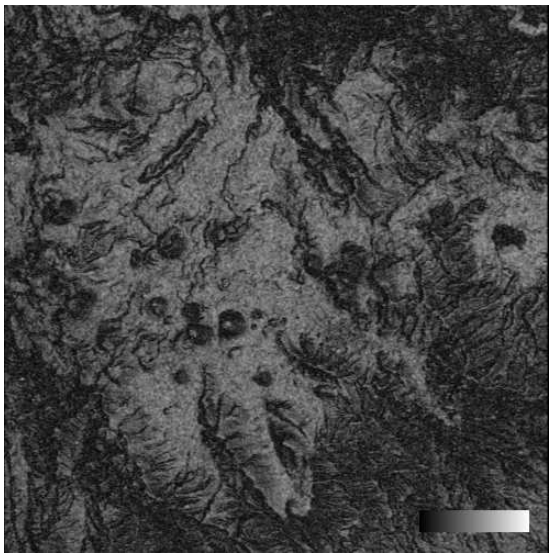

Figure 28.1: Coherence estimate using the REFPHASE_ONLY (old) method. The colorscale ranges from 0 to 1.

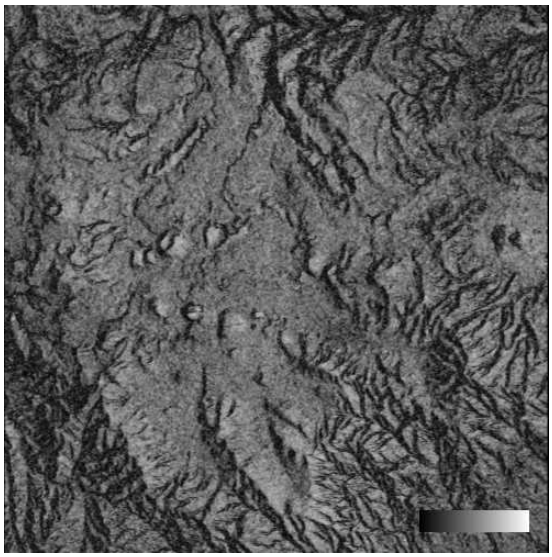

Figure 28.2: Coherence estimate using the INCLUDE_REFDEM (new) method. The colorscale ranges from 0 to 1.

Method selector for coherence map generation. INCLUDE_REFDEM computes the coherence map, with subtraction of both reference phase and radarcoded DEM phase, with the assumption that the last one has not been multilooked. By selecting the REFPHASE_ONLY method, the coherence map is computed with subtraction of the only reference phase (if available)

COH_OUT_CCOH *filename*
Filename of output datafile for complex coherence image (of step coherence). one of COH_OUT_CCOH and *_COH is mandatory.

COH_OUT_COH *filename*
Filename of output datafile for (real) coherence image (of step coherence). One of COH_OUT_CCOH and *_COH is mandatory.

COH_MULTILOOK *10 2*
Multilookfactor, if no multilooking is required, set this to ”1 1”.

COH_WINSIZE *10 2*
Window size of shifting window for coherence estimation.

Example input section for this step.

```
 c
 c
 comment ___product generation___
 c
COH_METHOD      include_refdem
 c COH_OUT_CCOH  Output/ccoh.raw                // complex image
COH_OUT_COH     Output/coh.raw                 // real
COH_MULTILOOK   10 2
COH_WINSIZE     10 2
```

28.2 Output Description

At successful exit the **process control flag** is switched on:

```
coherence:              1
```

Example output section for this step (in **products result file**).

```
***************************************************************
*_Start_coherence:
***************************************************************
Method:                                 INCLUDE_REFDEM
Data_output_file:                       42408_22735.coh
Data_output_format:                     real4
First_line (w.r.t. original_master):    3060
Last_line (w.r.t. original_master):     8052
First_pixel (w.r.t. original_master):   1719
Last_pixel (w.r.t. original_master):    2710
Multilookfactor_azimuth_direction:      5
Multilookfactor_range_direction:        1
Number of lines (multilooked):          998
Number of pixels (multilooked):         992
***************************************************************
* End_coherence:_NORMAL
***************************************************************
```

The output data file must be viewed with an external package like Matlab for now.

28.3 Implementation

The images are read in buffers for memory considerations. First complex interferogram is computed as in INTERFERO. and the norms of the master and slave images are computed.

Then a shifting window of size COH_WINSIZE is used to estimate the complex coherence (see Annex D).

The coherence is computed with a function of the matrix class. This function returns only the lines of the input which can be computed due to the edge of the the estimator window. Then this is multilooked (requires number of lines to be a multiple of the multilook factor)

Therefore, the buffer should contain an overlap with the previous one.

Chapter 29

FILTPHASE

This chapter describes the processing of step FILTPHASE. This step can be *optionally* used to filter the (latest) complex interferogram, in order to reduce noise, e.g., for visualization or aiding the phase unwrapping. It is probably best run after step SUBTRREFPHA. A lot of warnings can be generated if an image containing a lot of zeros is processed. These warnings can be ignored.

The method *goldstein* is described in [Goldstein and Werner, 1998]. Basically the fringes become sharper after filtering because the peak in the spectrum (caused by the fringes) is given a higher relative weight. Method *spatialconv* simply is a spatial convolution with a certain kernel function, e.g., a 3 point moving average. Method *spectral* is a multiplication of the spectrum with the kernel specified in an **input file** (e.g. a spectral low pass filter (LPF)).

29.1 Input Cards

PF_METHOD *goldstein*
Select goldstein method (”goldstein”), or spatial convolution (”spatialconv”) with cards PF_KERNEL and PF_IN_KERNEL2D, or spectral filter (”spectral”) with cards PF_IN_KERNEL2D, PF_BLOCKSIZE and PF_OVERLAP. For more info see implementation section.

PF_OUT_FILE *cint.alpha.filtered*
Output filename for complex real4 file with filtered phase (goldstein filter) where alpha is substituted. For method spatialconv, default is ”cint.filtered”

PF_IN_FILE *filename numlines*
Optional filename of complex real4 inteferogram (mph) file to be filtered instead of the default (which is obtained by reading the ’products’ **result files**). Also specify the number of lines in this file as second argument. This card is included to be able to filter files without having to create dummy **result files** to ’trick’ Doris. For now, the interferogram to be filtered must be complex real4.

PF_ALPHA *0.2*
This card is for method goldstein only. Alpha parameter for filtering. This parameter must be between 0 (no filtering) and 1 (most filtering). The card PF_KERNEL influences this value, since a higher smoothing, relative decreases the peak, and thus the effect of alpha.

PF_OVERLAP *3*

This card is for method *goldstein* and *spectral* only. Half of the size of the overlap between consecutive blocks and buffers, so that partially the same data is used for filtering. The total overlap should be smaller than PF_BLOCKSIZE. If this parameter is set to BLOCKSIZE/2-1 (the maximum value for this parameter) then each output pixel is filtered based on the spectrum that is centered around it. This is probably the best, but may be time consuming.

PF_BLOCKSIZE *32*

This card is for method *goldstein* and *spectral* only. Size of the blocks that are filtered. This must be a power of 2. It should be large enough so that the spectrum can be estimated, and small enough that it contains a peak frequency (1 trend in phase). (32 is recommended.))

PF_KERNEL *3 1/3 1/3 1/3*

This card is for method goldstein and spatialconv only. 1D Kernel function to perform convolution. First the number of elements in the kernel is given, then the values. The kernel is always normalized to 1 by dividing the kernel by the sum of the absolute values of the kernel.

For method goldstein: defaults to kernel [1 2 3 2 1]. This kernel is used to smooth the amplitude of the spectrum of the complex interferogram. The spectrum is later scaled by the smoothed spectrum to the power alpha.

For method spatialconv: Default is a 3 point moving average [1 1 1] convolution. The real and imaginary part is averaged seperately this way. For more info see implementation section. The output matrix has a zero valued edge of size floor(kernel/2).

PF_IN_KERNEL2D *filename*

This card is for method *spatialconv* and *spectral* only. Name of ascii **input file** to specify a 2D spatial kernel function. This file must start with a 1 line header containing numlines, numcols, scalingfactor. The next numlines lines contain the filter. numlines and numcols must be odd (centered) for method *spatialconv*. For method *spectral* they may be even; the zero frequency is located at position kernelsize/2-1 (starting at 0). The values of the kernel are multiplied by the scale factor. The kernel is not normalized in any other way. The output matrix has a zero valued edge of size floor(kernel/2).

Example of the cards for this step:

```
 c
 c
 comment   ___PHASEFILT___
 c
 c PF_METHOD    spectral
 c PF_IN_KERNEL2D  /proto/myfilter
 c PF_BLOCKSIZE 32
 c PF_OVERLAP   4
 c
 c PF_METHOD    spatialconv
 c PF_KERNEL    5 1 1 1 1 1
 c PF_IN_KERNEL2D  /proto/myfilter
 c
 c PF_METHOD    goldstein
PF_IN_FILE   Outdata/cint.srp.raw 323
PF_ALPHA     0.5
PF_KERNEL    5 1 1 1 1 1
PF_OVERLAP   4
PF_BLOCKSIZE 32
```

A simple example of PF_IN_KERNEL2D in an ascii **input file** (use this example with method *spatialconv*)

```
 5  5  0.05
```

```
 0  1  1  1  1
-1  0  1  1  1
-1 -1  0  1  1
-1 -1 -1  0  1
-1 -1 -1 -1  0
```

A second example of PF_IN_KERNEL2D ascii **input file** One could use cards: PF_BLOCKSIZE 32, and PF_OVERLAP 4, PF_METHOD spatial

```
15  15  1.0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The idea is to have a template directory with ascii kernels in it, that then can be used in doris. I do not have much experience with this.

One could generate the different filters with Matlab. For spectral method, one may want to filter with an Hamming window.

If you have matlab, paste the following to your terminal (uses Matlab) to generate the ascii file (just an example, please experiment yourself). Moreover, if you want to offer your ascii kernels for standard distribution of doris, please email it to us.

```
matlab << __EOFHD > /dev/null
SIZE = 32;
filterfile = ’filter.hamming’;
f = (standing(hamming(SIZE))*ones(1,SIZE)) .* ...
    (ones(SIZE,1)*lying(hamming(SIZE)));
fid = fopen(filterfile,’w’);
fprintf(fid,’%i %i 1.0\n’,SIZE,SIZE);
for ii=1:SIZE
  fprintf(fid,’%4.2f ’,f(ii,:));
  fprintf(fid,’\n’);
end
exit;
__EOFHD
```

And in Doris use the cards (using filter in spectral domain in this case):

```
PF_METHOD    spectral
PF_IN_KERNEL2D  filter.hamming
PF_BLOCKSIZE 32
PF_OVERLAP   4
```

29.2 Output Description

The **process control flag** for this step is switched on in the **products result file**:

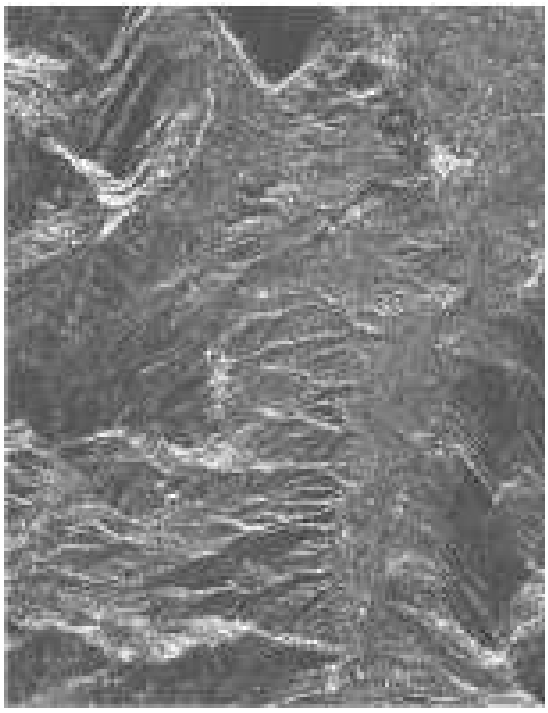

Figure 29.1: Magnitude of unfiltered complex interferogram.

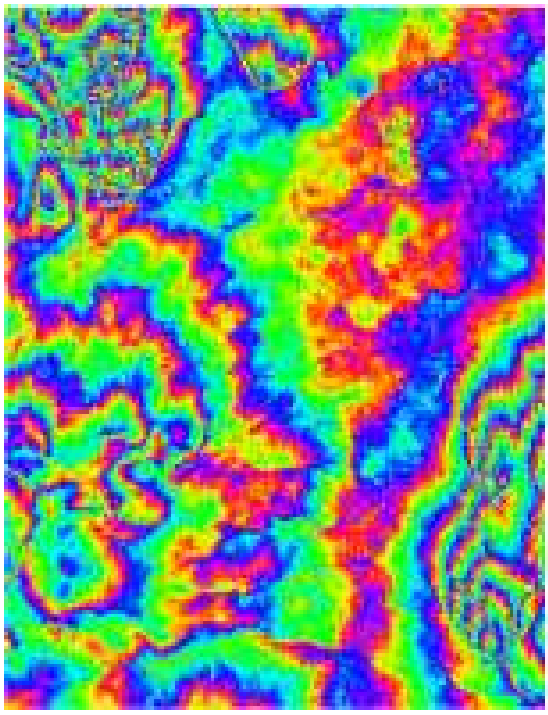

Figure 29.2: Phase of unfiltered complex interferogram.

```
filtphase:              1
```

And in the same **result file** a section will be added like (except if PF_IN_FILE is specified):

```
*********************************************************************
*_Start_filtphase:
*********************************************************************
Method: goldstein: size, alpha, overlap:        32 0.5 4
Input_file:                             Outdata/cint.srp.raw
Data_output_file:                       cint.0.5gf
Data_output_format:                     complex_real4
First_line (w.r.t. original_master):    1073
Last_line (w.r.t. original_master):     4302
First_pixel (w.r.t. original_master):   148
Last_pixel (w.r.t. original_master):    985
Multilookfactor_azimuth_direction:      10
Multilookfactor_range_direction:        2
Number of lines (multilooked):          323
Number of pixels (multilooked):         419
*********************************************************************
* End_filtphase:_NORMAL
*********************************************************************
```

29.3 Implementation

29.3.1 spatialconv

The complex interferogram is convoluted with a kernel by FFT's. The card PF_KERNEL specifies the 1D kernel. The 2D kernel is computed as: PF_KERNEL^T PF_KERNEL, e.g. for a 3 point moving average 1D

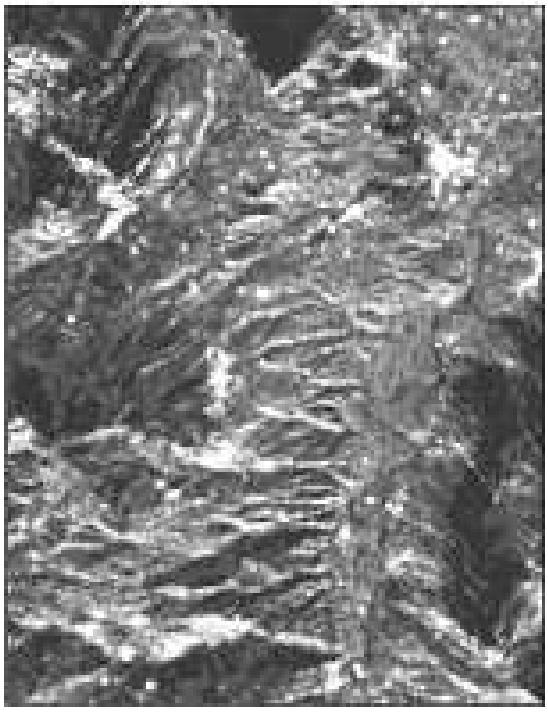

Figure 29.3: Magnitude of filtered complex interferogram. (Method: spatialconv.) A spatial convolution with a kernel [1 4 9 4 1] was used. Clearly a lot of detail is lost.

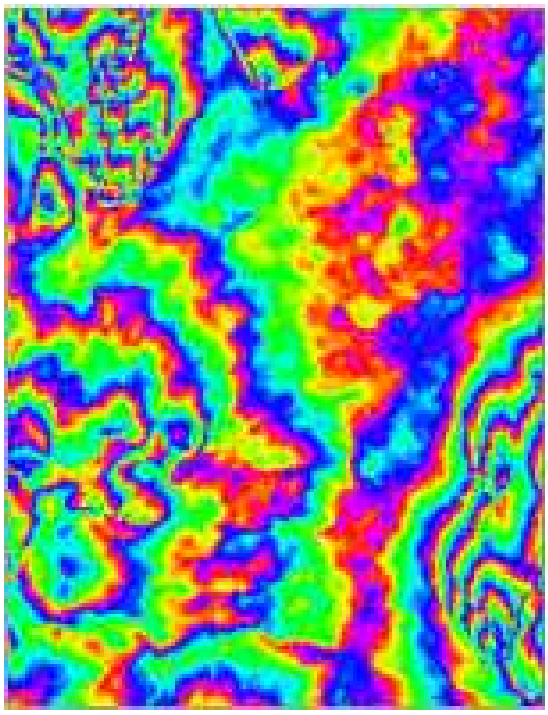

Figure 29.4: Phase of filtered complex interferogram. (Method: spatialconv.) A spatial convolution with a kernel [1 4 9 4 1] was used. Clearly a lot of detail is lost.

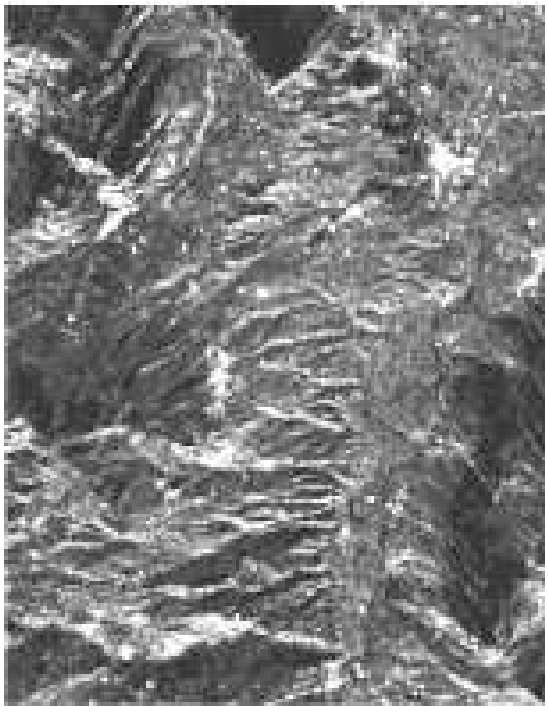

Figure 29.5: Magnitude of filtered complex interferogram. (Method: spectral.) A pointwise multiplication in the spectral domain by a 32 point hamming filter was used, a blocksize of 32, and an overlap of 4.

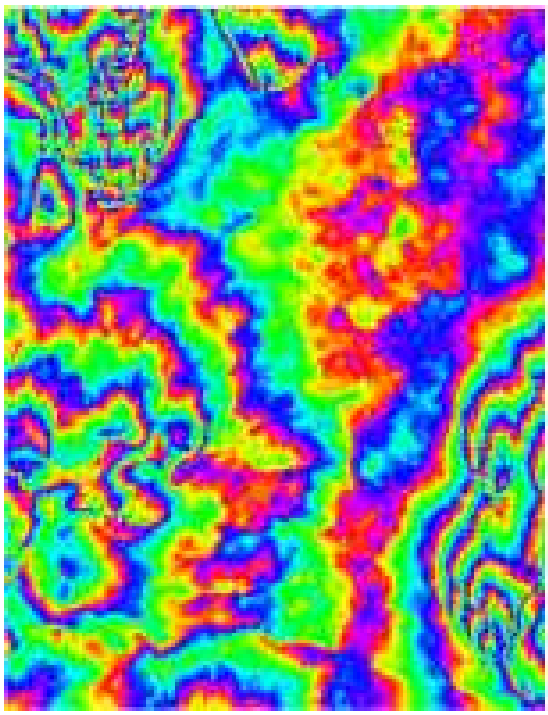

Figure 29.6: Phase of filtered complex interferogram. (Method: spectral.) A pointwise multiplication in the spectral domain by a 32 point hamming filter was used, a blocksize of 32, and an overlap of 4.

Figure 29.7: Magnitude of filtered complex interferogram. (Method: goldstein.) parameters used are alpha=0.5, smooth=3, overlap=4. This filter seems to preserve most detail.

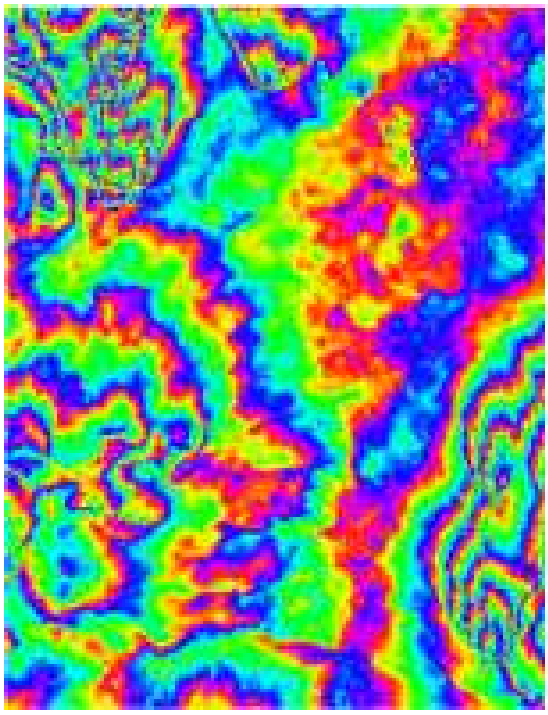

Figure 29.8: Phase of filtered complex interferogram. (Method: goldstein.) parameters used are alpha=0.5, smooth=3, overlap=4. This filter seems to preserve most detail.

kernel

$$\frac{1}{3}[111] \tag{29.1}$$

This becomes

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{29.2}$$

The blocksize for the convolution is chosen as high as possible. A 2D kernel can be specified in an **input file**. Only odd sized kernels can be used, but simply add a zero to an odd kernel.

If a real4 matrix containing phase should be convoluted by a certain kernel, first convert this real4 to a complex real4 matrix. Do this either by computing the phase for complex umbers with amplitude 1, or by setting the real part of the file to the phase and the imaginary part to 1 (arbitrary).

29.3.2 spectral

This method is implemented the same as goldstein method w.r.t. the overlap, blocksize etc. Algorithm per block (SIZE,SIZE) is to perform a 2D FFT of the block, and then to multiply pointwise with the kernel, which is padded with zeros. The kernel is centered around zero frequency.

29.3.3 goldstein

The algorithm is implemented as:

- Read in buffer B_i of PF_BLOCKSIZE lines (overlap).
- Get block B=B_ij as input block, see Fig. 29.9.
- B=fft2d(B) (obtain complex spectrum)
- A=abs(B) (magnitude of spectrum)
- S=smooth(A) (convolution with kernel)
- S=S/max(S) (S between 0 and 1)
- $B = B \cdot (S)^{\alpha}$ (weight complex spectrum)
- B=ifft2d(B) (result in space domain)
- If all blocks of buffer done, write to disk.

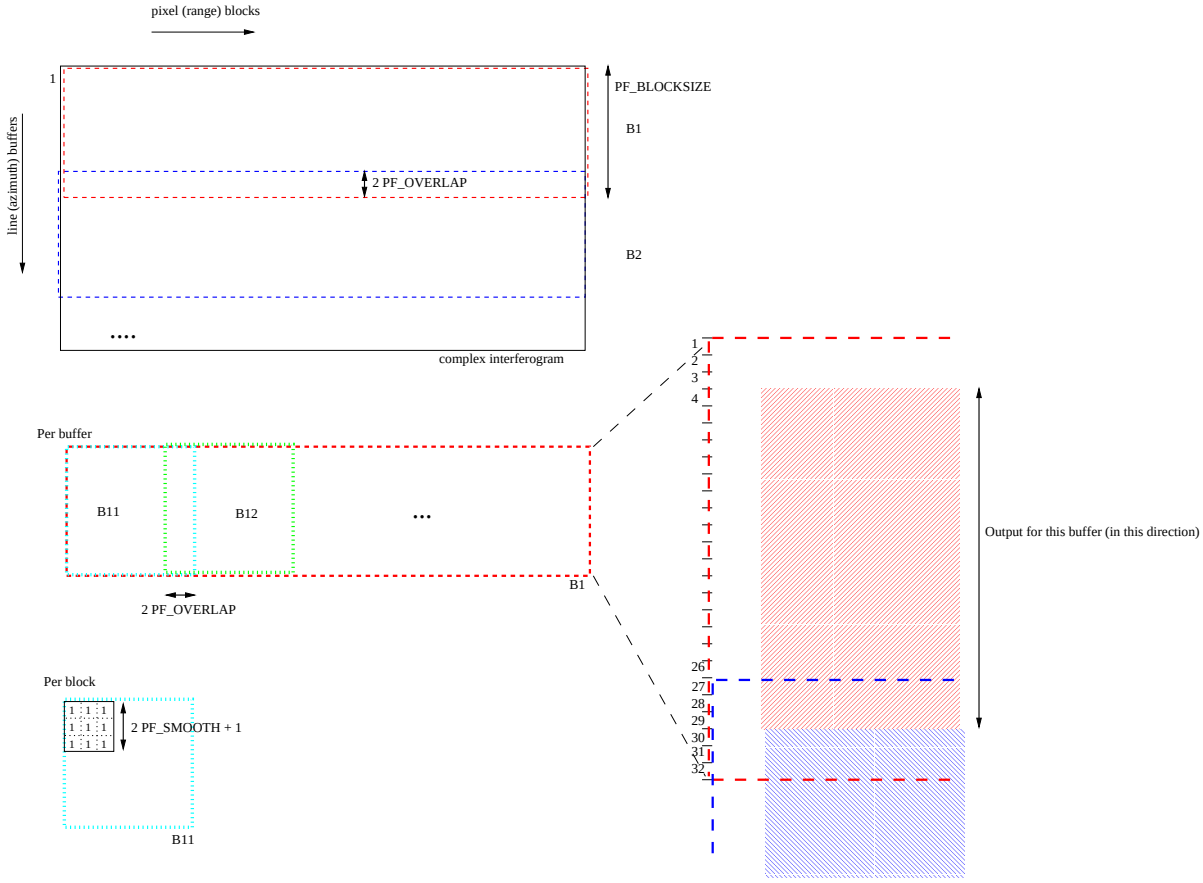

Figure 29.9: Buffering of complex interferogram in blocks for phase filtering.

For a block [pixlo:pixhi], e.g., [0:15], the output equals for an overlap (=3), [pixlo+overlap:pixhi-overlap], [3:12]. The number of output equals size-2overlap = pixhi-pixlo+1-2overlap = 10.

Chapter 30

UNWRAP

In this chapter the processing of step UNWRAP is described. This step is currently not implemented within the Doris software. To obtain the unwrapped interferogram, you should use another software, for example one of the routines of [Ghiglia and Pritt, 1998] which can be obtained by ftp at ftp.wiley.com/public/sci_tech_med/phase_unwrapping. These software should not be considered public domain, you ought to buy the book. The slant to height conversion and geocoding can only be done with an unwrapped interferogram.

Recently ”snaphu” of Curtis Cheng was put in the public domain. It is recommended you install this software as standalone executable, and continue with Doris for geocoding afterwards. METHOD snaphu can be used from within Doris. But experience has to be gained how this software best performs.

Sometimes the coherence as computed by doris seems to contian NaNs (not-a-number). snaphu does not expect this and exits when this happens. In Matlab the created coherence file can be easily corrected with, e.g.,

```
q=freadbk('9192_6687.coh',2577,'float32');
idxx=isnan(q);
idx=where(idxx==1);
q(idx)=0.0001;
fwritebk(q,'coh_no_nan','float32');
```

If you use a standalone application to unwrap the interferogram, you might have to mimic the output as described below, so Doris can obtain the current filename and dimensions for the unwrapped interferogram from the interferogram **result file**.

30.1 Input Cards

For the snaphu program, please also refer to their website and read the man page. Doris is a wrapper for a system call to the executable snaphu. Therefore, a program called snaphu should be executable and in your path. An input file for snaphu is created in the current directory. You can rerun snaphu with a changed inputfile from the prompt if required (but keep same output file name, format as it is in the **result file** for the interferogram.) We assume to unwrap the complex interferogram, mph format always for snaphu.

UW_METHOD *SNAPHU* | *TREEFRAMON*
Select method for unwrapping. For general users, if they have installed the Snaphu program, this will make a system call. Other methods are not available fur public domain.

UW_OUT_FILE *uint.hgt*
Output filename of unwrapped interferogram.

UW_OUT_FORMAT *HGT* | *REAL4*
Output format of unwrapped interferogram.

UW_SNAPHU_MODE *TOPO* | *DEFO* | *SMOOTH* | *NOSTATCOSTS*
Output format of unwrapped interferogram. Snaphu options -t, -d, -s respectively. Refer to snaphu manual for more information.

UW_SNAPHU_COH *filename*
use specified file for correlation values. This file must be registered and have the same number of looks. Snaphu option -c. Refer to snaphu manual for more information.

UW_SNAPHU_LOG *filename*
Output log file name for snaphu option -l. Refer to snaphu manual for more information.

UW_SNAPHU_INIT *MST* | *MCF*
Output log file name for snaphu option -l. Refer to snaphu manual for more information.

UW_SNAPHU_VERBOSE *ON* | *OFF*
snaphu option -v. Refer to snaphu manual for more information.

30.2 Output Description

At successful exit the **process control flag** is switched:

```
unwrap:                 1
```

The section for the unwrapping in the **result file** for the interferogram looks like the following for method TREEF (file name and format are used later):

```
*********************************************************************
*_Start_unwrap:
*********************************************************************
Data_output_file:                       Outdata/uint.raw
Data_output_format:                     real4
Data_output_file_regions:               Outdata/regions.raw
Data_output_format:                     short int (2B)
First_line (w.r.t. original_master):    1001
Last_line (w.r.t. original_master):     2105
First_pixel (w.r.t. original_master):   501
Last_pixel (w.r.t. original_master):    700
Multilookfactor_azimuth_direction:      10
Multilookfactor_range_direction:        2
Program for unwrapping:                 treef_ramon
Output program for unwrapping:          ramon.uw
Delta lines for seed:                   100
Delta pixels for seed:                  100
Number of patches used:                 1
*********************************************************************
* End_unwrap:_NORMAL
*********************************************************************
```

If the 'Data_output_format:' is 'real4', then the output is assumed to be real4 unwrapped phase values. If the unwrapping was not successful, these pixels are set to -999. and ignored for slantrange to height conversion and differential insar.

The ’Data_output_format:’ of the unwrapped interferogram also can be ’hgt’, a band interleaved format (amplitude, phase) for SNAPHU. For more details on the definition if unwrapping went wrong, see Annex D.

Chapter 31

DINSAR

This chapter describes the processing step DINSAR, which stands for (3 or 4 pass) differential interferometry.

Three pass differential interferometry is described in [Zebker et al., 1994]. It is a method to remove the topographic induced phase from an interferogram containing topography, deformation, and atmosphere. This module thus can also be used to study atmospheric effects in interferograms, if no deformation is expected.

This step can be performed if an unwrapped topography interferogram (topo pair) and a complex deformation interferogram (defo pair) are present (with a common master, perform 2 seperate runs of Doris to achieve this). The interferograms have to be corrected for the phase of the 'flatearth' (see step SUBTRREFPHA), and sampled on the same grid (see step RESAMPLE). The files must have the same multilook factors and the same dimensions (i.e. overlap exactly). The perpendicular baseline of the topo-pair should be larger than that of the defo-pair, to prevent that noise is blown up, but this cannot always be controlled of course.

This step is performed in the defo-pair processing tree. First create a directory to run the topo-pair processing until a unwrapped interferogram is obtained (keep the master, slave, and products **result files**). Then perform the defo-pair processing. After interferogram generation and 'flatearth' subtraction, start this step (DINSAR), specifying the location of the **result files** of the topo-pair processing with input cards. For 3 pass, use a common master. For 4 pass, coregister the complex interferogram on the complex interferogram of the defo-pair, and then unwrap, or first coregister master and slave on the master of the deformation pair.

To geocode the differential phase values, geocode the topo-interferogram and use the latitude/longitude matrices for the differential grid.

31.1 Input Cards

DI_OUT_FILE *differentialinterf.raw*
Output filename for complex real4 file with differential phase (in slant range system).

DI_IN_TOPOMASTER *same as* **master result file** *card*
Specify this card if **4 pass** differential interferometry is required. Do not use this card for 3 pass, or use the same name as the **master result file** for the defo pair (see chapter 2). Filename of the master of the topo-pair. To obtain the orbit and other parameters for the topo master.

DI_IN_TOPOSLAVE *result file name*
Filename of the slave of the topo-pair. To obtain the orbit and other parameters for the primary slave.

DI_IN_TOPOINT *result file name*

Filename of the interferogram **result file** of the topo-pair. To obtain the name and dimensions of the unwrapped (topography) interferogram.

DI_OUT_SCALED *filename*
Filename for optional (debug) output of a real4 file with scaled (with ratio of perpendicular baselines) unwrapped topo interferogram.

Example of the cards for this step:

```
 c
 c
 comment   ___ DINSAR 3 PASS ___
 c
DI_OUT_FILE         ./Outdata/difg.raw
 c DI_IN_TOPOMASTER  /data/project/topo/master.res // if 4 pass method
DI_IN_TOPOSLAVE      /data/project/topo/slave.res
DI_IN_TOPOINT        /data/project/topo/products.res
 c DI_OUT_SCALED    ./Outdata/scaled.raw           // debug
```

31.2 Output Description

In the defo processing **result file** for the products, the **process control flag** for dinsar is switched on.

```
dinsar:                1
```

A complex real4 ('mph') file is created with the wrapped differential phase. (The amplitude is the same as that of the original 'deformation' interferogram). A complex value (0,0) indicates unwrapping was not ok.

If the the debug version of Doris is used (compiled with __DEBUG) then ascii matrices are dumped for Linenumber, Pixelnumber, Bperptopo, Bperpdefo, and Ratio.

Figures 31.1, 31.2, and 31.3 give example of output.

31.3 Implementation

See also [Zebker et al., 1994]. See figure 31.4

Simple equations for topo-pair (no deformation, no atmosphere, no other errors, $r_1 \parallel r_2$)

$$\theta = \theta_0 + \delta\theta \tag{31.1}$$

$$B_{\parallel} = r_1 - r_2 \tag{31.2}$$

$$B_{\perp} = B\cos(\theta - \alpha) = B\cos(\alpha - \theta) \tag{31.3}$$

$$B_{\parallel} = B\sin(\theta - \alpha) = -B\sin(\alpha - \theta) \tag{31.4}$$

The baseline components, for points on the reference ellipsoid ($h = 0$) are

$$[B_{\perp}]_{h=0} = B_{\perp 0} = B\cos(\theta_0 - \alpha) \tag{31.5}$$

$$\left[B_{\parallel}\right]_{h=0} = B_{\parallel 0} = B\sin(\theta_0 - \alpha) \tag{31.6}$$

The 'true' phase of the interferogram is

$$\phi = -\frac{4\pi}{\lambda} B_{\parallel} \tag{31.7}$$

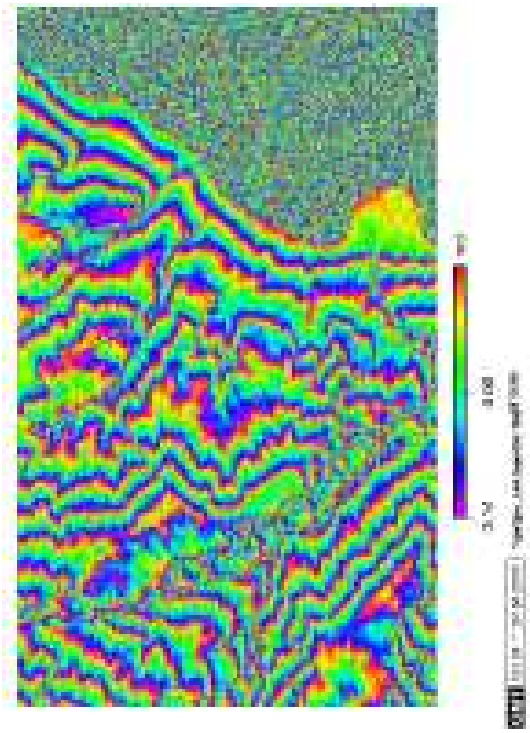

Figure 31.1: Phase of complex 'topography' interferogram (flat earth corrected, cropped). The area is dead sea Israel. Temporal baseline is 1 day (tandem). The perpendicular baseline is approximately 105 meters. This interferogram has been coregistered on the defo pair (31.2) by tricking Doris.

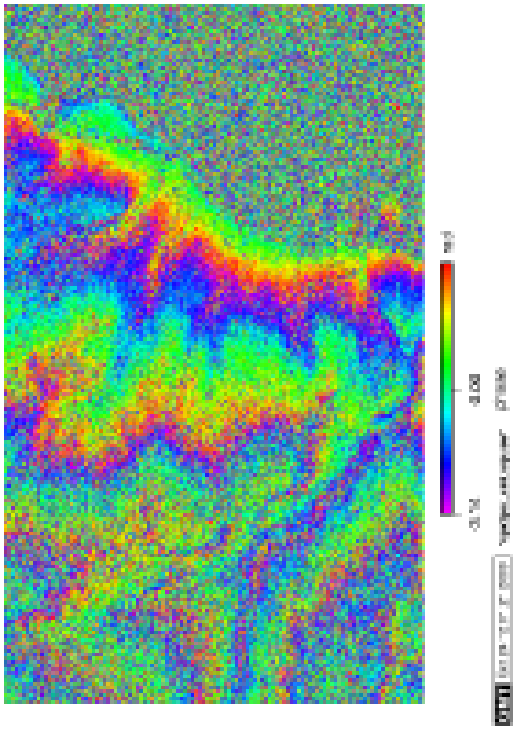

Figure 31.2: Phase of complex 'deformation' interferogram (flat earth corrected, cropped). The area is dead sea Israel. Temporal baseline is 28 months day. The perpendicular baseline is approximately -30 meters.

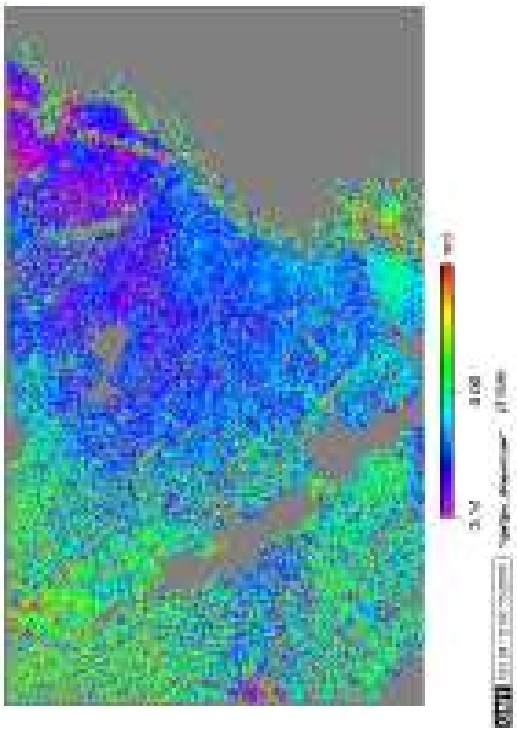

Figure 31.3: Phase of differential complex interferogram (result of step DINSAR, cropped). The area is dead sea Israel. The topography is removed from the original interferogram (Figure 31.2) by scaling the topography interferogram (Figure 31.1). The perpendicular baseline is approximately 30 meters.

And corrected for the phase of the reference body

$$\phi = -\frac{4\pi}{\lambda}(B_{\parallel} - B_{\parallel 0}) \tag{31.8}$$

For the defo-pair (1,3), denoted with a prime, similar equations follow. Deformation in the line of sight (range), that occurred in between the acquisitions, is denoted by Δr

$$\Delta r = -\frac{\lambda}{4\pi}\phi_{\Delta r} \tag{31.9}$$

A positive Δr implies deformation in the $B_{\parallel}$ direction (away from the sensor, i.e., subsidence). The phase of this interferogram is

$$\phi' = -\frac{4\pi}{\lambda}(r_1 - (r_3 + \Delta r)) = -\frac{4\pi}{\lambda}({B_{\parallel}}' + \Delta r) \tag{31.10}$$

Combining the expressions for the interferometric phase for the topo-pair (31.7) and defo-pair (31.10) yields:

$$\phi' = \phi\frac{{B_{\parallel}}'}{B_{\parallel}} + \frac{4\pi}{\lambda}\Delta r \tag{31.11}$$

The problem here is that the 'true' parallel baselines are unknown.

The (actually wrapped) phase of the deformation interferogram, corrected for reference phase, is defined as:

$$\begin{aligned} \phi' &= -\frac{4\pi}{\lambda}[{B_{\parallel}}' - {B_{\parallel 0}}' + \Delta r] \qquad (31.12)\\ &= -\frac{4\pi}{\lambda}[B'\sin(\theta - \alpha') - B'\sin(\theta_0 - \alpha') + \Delta r] \\ &= -\frac{4\pi}{\lambda}[B'\sin(\beta' + \delta\theta) - B'\sin\beta' + \Delta r] \end{aligned}$$

where $\beta' = \theta_0 - \alpha'$. Using the approximation for small $\delta\theta$ (which is about 1° or 0.0175 rad for terrain height differences of 5 km)

$$\sin(\beta + \delta\theta) = \sin\beta\cos\delta\theta + \cos\beta\sin\delta\theta \approx \sin\beta + \delta\theta\cos\beta \tag{31.13}$$

it follows from equation 31.13 that the 'flat earth' corrected phase equals

$$\begin{aligned} \phi' &= -\frac{4\pi}{\lambda}[B'(\sin\beta' + \delta\theta\cos\beta') - B'\sin\beta' + \Delta r] \\ &= -\frac{4\pi}{\lambda}[\delta\theta B'\cos\beta' + \Delta r] \\ &= -\frac{4\pi}{\lambda}\delta\theta {B_{\perp 0}}' - \frac{4\pi}{\lambda}\Delta r \end{aligned} \tag{31.14}$$

The corrected phase for the topo pair equals $\phi = -\frac{4\pi}{\lambda}\delta\theta B_{\perp 0}$ (using the same approximation), and combining this with 31.15 yields (for the 'flat earth' corrected phases)

$$\phi' = \phi\frac{\delta\theta {B_{\perp 0}}'}{\delta\theta B_{\perp 0}} - \frac{4\pi}{\lambda}\Delta r = \phi\frac{{B_{\perp 0}}'}{B_{\perp 0}} - \frac{4\pi}{\lambda}\Delta r \tag{31.15}$$

or

$$\Delta r = -\frac{\lambda}{4\pi}[\phi' - \phi\frac{{B_{\perp 0}}'}{B_{\perp 0}}] \tag{31.16}$$

or for the phase $\phi_{\Delta r}$ caused by the deformation Δr

$$\phi_{\Delta r} = \phi' - \frac{{B_{\perp 0}}'}{B_{\perp 0}}\phi \tag{31.17}$$

This important equation shows how to obtain offset vectors from 3 SLC images, i.e., by scaling the (reference phase corrected) unwrapped phase of the topo-pair by the ratio of the perpendicular baselines (to points on reference body), and subtracting this from the phase of the defo-pair. This can thus be performed without the 'true' values for θ are required.

31.3.1 Algorithm

Input is the unwrapped topo-interferogram (corrected for 'flat earth'). Format is hgt, or real4. Not unwrapped thus indicated by NaN==-999. (real4) or if amplitude==0 (hgt). Defo-interferogram is wrapped (complex real4, mph) (specified in interferogram **result file**).

1. Obtain orbit for topo-slave (**result file**). Obtain filename/dimension of unwrapped interferogram (**result file**).
2. Read in matrices, appropriate size/format etc. per line. Check if they exactly overlap.
3. Compute $B_\perp$ and ${B_\perp}'$ on a small grid (20x10 points over the image).
4. Model the ratio of the perpendicular baselines by a 2D polynomial of degree 1. (r(l,p) = a00 + a10l + a01p) Give statistics for max. error due to modelling. (It seems the ratio hardly changes over the image for ERS1/2).
5. Compute wrapped deformation phase (phase corrected for topography) with formula 31.17, using the modeled ratio r_{ij}. (Actually compute it complex: $c_{ij}* = \cos(r_{ij}\cdot\phi) - i\sin(r_{ij}\cdot\phi)$.)
6. Set not unwrapped regions to (0.,0.)
7. Write **output file** (complex real4, mph format). (if a problem with unwrapping occurred, write (0,0).) If requested, also write the scaled unwrapped interferogram.

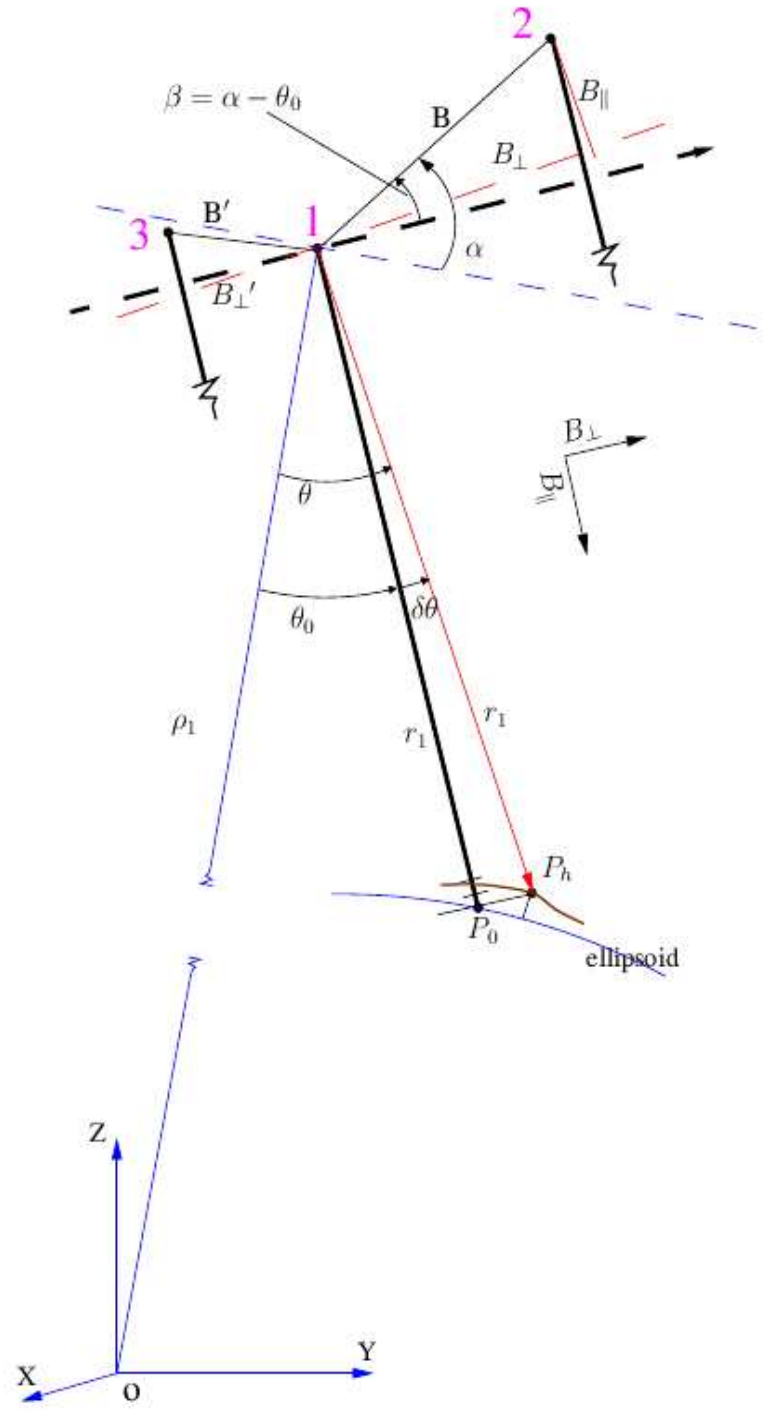

Figure 31.4: Geometric configuration for 3-pass differential insar. The orbits go 'into' the paper. All angles are defined counterclockwise. The terrain element P corresponding to the radar coordinate (l,p) is located at a height h above the ellipsoid. The perpendicular baseline required for this method is the one for points P located on the reference ellipsoid ($h = 0$). $\delta\theta$, the change in θ since P is on a height h, due to a 5 km height difference, is approximately 1°.

Chapter 32

SLANT2H

In this chapter the processing of step SLANT2H is described. In this step in principle the heights in the radar coded system are computed. However with the exact method, the geocoding can be done in the same step.

The results of the three implemented methods are different, so a comparison has been made.

Processing is in buffers for all methods, while it is possible just to do it line by line. In case a polynomial has to be evaluated (rodriguez method) it is more efficient to have a buffer.

32.1 Input Cards

S2H_METHOD *ambiguity* | *schwabisch* | *rodriguez*
Method selector. ambiguity geocodes as well, uses height ambiguity to compute the height. schwabisch method uses polynomials to compare the phase with the reference phase. rodriguez method uses geometry, contains an approximation, it is not clear how to compute a certain parameter.

S2H_OUT_HEI *hei.raw*
Output file name for computed height values.

S2H_OUT_PHI *phi.raw*
Only for ambiguity method. Output file name for computed phi values (latitude).

S2H_OUT_LAM *lam.raw*
Only for ambiguity method. Output file name for computed lambda values (longitude).

S2H_NPOINTS *200*
Only for schwabisch method. the number of locations to compute the reference phase at different altitudes.

S2H_DEGREE1D *2*
Only for schwabisch method. the degree of the 1d polynomial to fit reference phase through at every location.

S2H_NHEIGHTS *s2h_degree1d+1*
Only for schwabisch method. the number of heights to evaluate the reference phase. minimum is default.

S2H_DEGREE2D *5*

Only for schwabisch method. the degree of the 2d polynomial to fit 1d coefficients as function of location.

Example input:

```
 c
 c
 comment   ___ SLANT 2 HEIGHT CONVERSION ___
 c
 S2H_METHOD     schwabisch
 S2H_NPOINTS    500
 S2H_DEGREE1D   2
 S2H_NHEIGHTS   3
 S2H_DEGREE2D   5
 S2H_OUT_HEI    Outdata/hei.schw
c
c S2H_METHOD    ambiguity
c S2H_OUT_HEI   Outdata/hei.ambi
c S2H_OUT_LAM   Outdata/lam.ambi
c S2H_OUT_PHI   Outdata/phi.ambi
c
c S2H_METHOD    rodriguez
c S2H_OUT_HEI   Outdata/hei.rodr
```

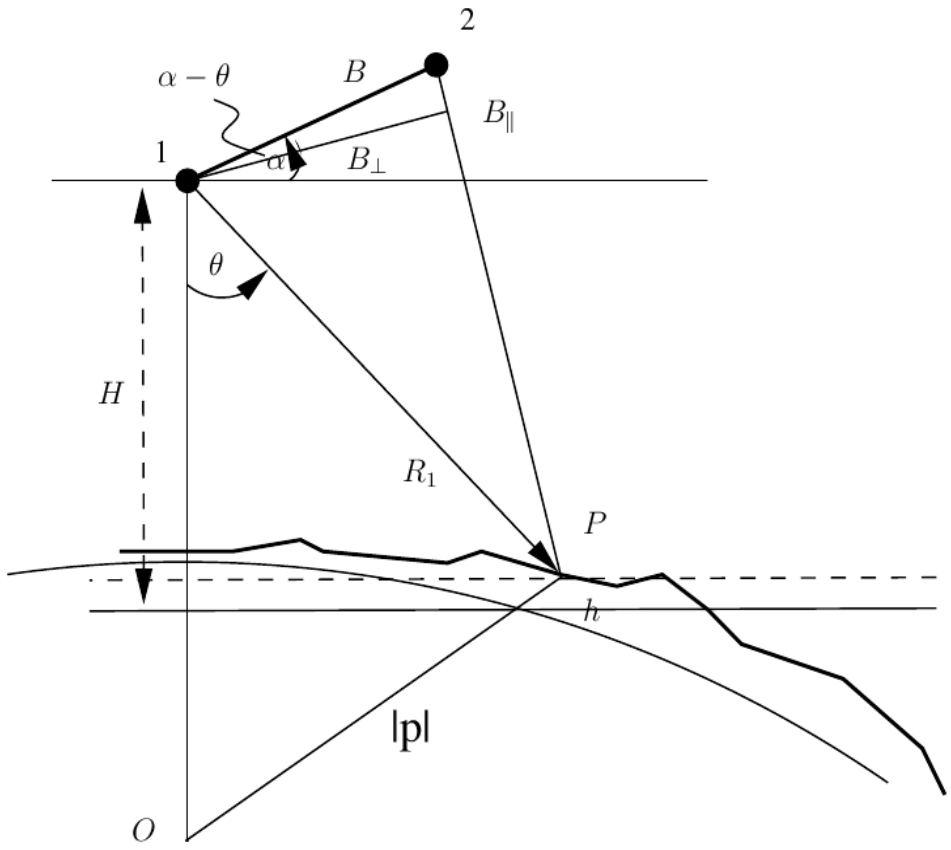

Figure 32.1: Geometric configuration for slant to height conversion.

32.2 Output Description

The **process control flag** is switched at successful exit:

```
slant2height:           1
```

An example of the output in the **products result file**:

```
*****************************************************************
*_Start_slant2h
*****************************************************************
Method:              schwabisch
Data_output_file:                       Outdata/hei.schwabisch
Data_output_format:                     real4
First_line (w.r.t. original_master):    1001
Last_line (w.r.t. original_master):     2105
First_pixel (w.r.t. original_master):   501
Last_pixel (w.r.t. original_master):    700
Multilookfactor_azimuth_direction:      10
Multilookfactor_range_direction:        2
Ellipsoid (name,a,b):                   WGS84 6.37814e+06 6.35675e+06
*****************************************************************
* End_slant2h:_NORMAL
*****************************************************************
```

In the **output files** the height is stored. (number of lines etc. multilooked unwrapped interferogram.) etc.

32.3 Implementation

32.3.1 Method ambiguity

This method yields first the heights of the (line,pixel) and the position P(x,y,z). In this manner the geocoding can be done in the same step. by converting P(x,y,z) (known h) to phi,lambda. If there is a trend in the height this has to be removed first, e.g. by using tiepoints. This means the computed phi and lambda matrices are not correct anymore.

With the baseline defined as in Annex D the following equations hold.

$$B_{\|} = r_1 - r_2 \tag{32.1}$$

$$B_{\|} = B\sin(\theta - \alpha) \tag{32.2}$$

$$B_{\perp} = B\cos(\theta - \alpha) \tag{32.3}$$

Note the sign, must be minus.

$$\phi_i = -\frac{4\pi}{\lambda} r_i + \phi_{obj} \tag{32.4}$$

Content of unwrapped interferogram (never mind the -phiR).

$$\phi = \phi_1 - \phi_2 - \phi_R = -\frac{4\pi}{\lambda} B_{\|} - \phi_R \tag{32.5}$$

$$\frac{d\phi}{d\theta} = -\frac{4\pi}{\lambda}\frac{dB_{\|}}{d\theta} = -\frac{4\pi}{\lambda} B_{\perp} \tag{32.6}$$

Geometric equation:

$$h = H - r_1\cos\theta \tag{32.7}$$

$$\frac{dh}{d\theta} = r_1 \sin\theta \tag{32.8}$$

Height ambiguity:

$$\frac{dh}{d\phi} = \frac{dh}{d\theta}\frac{d\theta}{d\phi} = -\frac{\lambda}{4\pi}\frac{r_1 \sin\theta}{B_\perp} = -\frac{\lambda}{4\pi}\frac{r_1 \sin\theta}{B_h \cos\theta + B_v \sin\theta} \tag{32.9}$$

Final equation to convert phase to height:

$$h = -\frac{\lambda}{4\pi}\frac{r_1 \sin\theta}{B_h \cos\theta + B_v \sin\theta}\phi \tag{32.10}$$

The procedure to compute the height is as follows (note that computation is skipped if unwrapping went wrong, indicated by NaN (not a number) in the unwrapped interferogram.)

1. for all lines
   (a) .i1. compute B_h and B_v (by computing h for middle pixel, then compute baseline).
   (b) .i2. hcurrent = 0
   (c) for all pixels
      i. .j1. if value unwrapped phase equals NaN $\equiv$ -999 then goto next pixel
      ii. .j2. compute θ (corresponding to h)
      iii. .j3. hlast=hcurrent, compute hcurrent with formula
      iv. if ((hlast - hcurrent) $>$ k) then goto .j2.

32.3.2 Method rodriguez

Use same definitions as exact method, see also Annex D. See also [Rodriguez and Martin, 1992]. This method uses the geometry to compute $\sin(\theta - \alpha)$. There are two errors in it for now. First H is not computed exact. Second the baseline parameters are not computed exact per line. P is evaluated at reference surface and S according to that position. This means that if the orbits are not parallel the point S is not computed correctly, which introduces errors in the baseline computation. The co-registration model is better used for that.

Known:
r1 range to M,P
position M
B baseline
ξ baseline orientation with regard to (equals our def. of alpha)

Compute:
theta (angle state, look: with formulas exact (no iterations)
H height sat above some surface ?? how to compute this exact??

The following we derived for our baseline definition: (beta = angle (2-1,P-1) counterclockwise:)

$$\beta = \angle(2-1, P-1) \tag{32.11}$$

$$\cos\beta = \cos(\frac{1}{2}\pi + (\alpha - \theta)) \tag{32.12}$$

$$= \sin(\theta - \alpha) \tag{32.13}$$

$$(r_1 - B_\parallel)^2 = r_1^2 + B^2 - 2r_1 B\cos\beta \tag{32.14}$$

$$\sin(\theta - \alpha) = \frac{(r_1 - B_\parallel)^2 - r_1^2 - B^2}{-2Br_1} \tag{32.15}$$

$$B_\parallel = -\frac{\lambda}{4\pi}\phi \tag{32.16}$$

So theta can be solved for exact with these formulas. Note:

$$\theta - \alpha = \arcsin \frac{(r_1 - B_\parallel)^2 - r_1^2 - B^2}{-2Br_1} = x \tag{32.17}$$

$$\theta = \arcsin x + \alpha \quad \vee \quad \theta = \pi - \arcsin x + \alpha \tag{32.18}$$

I did not find an efficient way to always use the right expression yet. Now I use the fact that theta is about 20 degrees, but it should be possible to find out quadrants.

Compute H from known: theta, position Master (rho1), r1 In triangle (1,P,0) three Start with cosine law for line across theta = p

$$p^2 = \rho_1^2 + r_1^2 - 2rho_1 r_1 \cos\theta \tag{32.19}$$

Then compute cosine of angle mu across r1 in same triangle

$$r_1^2 = \rho_1^2 + p^2 - 2rho_1 p \cos\mu \tag{32.20}$$

Unclear how to compute H exact. for now use approximation. (set radius of earth at location of satellite equal to radius at location of P) compute satellite height by Bowring's method (xyz2ell) then radius R of earth at phi,lambda to satellite:

$$R = \rho_1 - H_{\text{sat}} \tag{32.21}$$

Approximate H in this way

$$H = \rho_1 - R\cos\mu \tag{32.22}$$

Compute error of this approximation (**preliminary study!**):

This will cause a bias and some trend in the height. Because there likely already is a trend due to orbit errors, this is not as bad as it might seem. By using tie points a good height may be computed. For now we did not implement a routine that uses tie points.

New way of computing H: (NOT implemented, to difficult)

1. compute in new system (x,y) coordinates of 1(0,rho1), P(..),...
2. ellips equation in tje same system, rotated over co-latitude
3. snijpunt P,ellips := R
4. H = rho1-Rq

Problems with this method: how do you know orientation of theta? rotation of ellips to new system.

A few more notes:

$$B_\parallel = -\frac{\lambda}{4\pi}(\phi + \phi_R) \tag{32.23}$$

So the reference phase has to be added again in order to compute Bpar. (otherwise Bpar is 0.001 m or so.)

Processing:

1. per line compute B,alpha;

2. per pixel
   (a) phi to Bpar
   (b) r known
   (c) compute theta (exact?)
   (d) compute p
   (e) compute mu
   (f) compute H
   (g) compute h

The idea is to compute H from the position of M. (H can be computed by (reference needed) as shown.) And then to find theta=f(B,phi,r). And then find h with the first equation. This method is implemented to check our exact method, the results are very different.

In this method the point P does not have to be computed. (though in order to compute baseline components we will compute a point P on height h (evaluate, iteratively) once for every line.)

A better way might be to use the co-registration model to compute the point S.

32.3.3 Method schwabisch

This method is described in [Schwäbisch, 1995]. It is a fast method that yields the radar coded heights. It is based on the idea to first compute the reference fase at a number of heights and then to compare the actual phase from the interferogram with these values to determine the height.

A problem is that the interferogram does not contain the reference phase anymore, so that has to be added to the estimated phi.

It uses a number of steps which are described below.

1. Compute reference phase at a NL locations (line,pixel) at NH (=3) heights (0, 2000 and 4000m).
   for h=0, 2000, 4000:
   - ellips.a=wgs84.a+h, ellips.b=wgs84.b+h
   - compute position of master satellite, corresponding point P on ellips and position of slave satellite, see annex D.
   - $B_{\|} = r_1 - r_2$
   - $\phi_R = -\frac{4\pi}{\lambda} B_{\|}$
   - store the values and locations and heights.

   Note that the reference phase defined like this typically is something like 5000 (rad) even for h=0. Therefor the reference phase for h=0 is set to 0, (because in the unwrapped interferogram the reference phase is removed, if the phase of the unwrapped interferogram is 0, then this should yield a height of 0) and the reference phase for height h is set to refphaseh - refphase0. (This makes the computations as done later a little stupid, to estimate coefficients which are by definition equal to 0.) (An other possibility is not to do this here, but later when the functions are evaluated to add the reference phase to each pixel. I have tested this and the results are identical (+- .4m))
2. Compute for each location a polynomial (1d degree 1dD (= NH-1)) to describe the height as a function of reference phase at these points.
   For each location it holds (ϕ_i for height i):

$$h = 0 = \alpha_0 + \alpha_1\phi_0 + \alpha_2\phi_0^2 \quad (32.24)$$

$$h = 2000 = \alpha_0 + \alpha_1\phi_1 + \alpha_2\phi_1^2 \quad (32.25)$$

$$h = 4000 = \alpha_0 + \alpha_1\phi_2 + \alpha_2\phi_2^2 \quad (32.26)$$

   So it is easy to solve (exact) for α_i per location.

3. Compute (1dD+1=NH) polynomials to describe the coefficients of the previous step as a function of location. (now, for a random location, the coefficients of height as a function of the reference phase can be computed.) For example for α_0 computed at NL locations (l,p) a 2d polynomial can be used:

$$\begin{aligned}\alpha_{0_{lp}} &= \beta_{00} + \beta_{10}l + \beta_{01}p + \beta 20 l^2 + ... \\ &= \sum_{i=0}^{d}\sum_{j=0}^{i}\beta_{i-j,j}l^{i-j}p^j \end{aligned} \tag{32.27}$$

   A linear system can be easily set up and solved (least squares) by cholesky factorization. A rescaling needs to be applied to avoid instability. The system can be solved simultaneously for all alphas, because the normal matrix (and factorization) remains the same, but somehow our cholesky routine introduced an error (which is probably caused by using fortran in c) so we just solve 3 seperate times with the same factored normalmatrix.

4. Evaluate for all points at (line,pixel) with ok unwrapped phase the 2d polynomial to obtain the coefficients of the height(phi) function. Then evaluate the 1d polynomial to obtain the height.
   For all (l,p) with ok unwrapped phase:

   - Compute the alphas.
     For betas appropriate to alpha0:

$$\alpha_0 = \sum_{i=0}^{d}\sum_{j=0}^{i}\beta_{i-j,j}l^{i-j}p^j \tag{32.28}$$

   - For betas appropriate to alpha1:

$$\alpha_1 = \sum_{i=0}^{d}\sum_{j=0}^{i}\beta_{i-j,j}l^{i-j}p^j \tag{32.29}$$

   - repeat computing alphas until you have them all (1dD+1).
   - Compute the height.

$$h = \sum_{i=0}^{1dD}\alpha_i\phi^i \tag{32.30}$$

32.4 Comparison of the methods

Here a simple test is described that was performed to see the differences between the methods. A unwrapped interferogram was obtained of the Veluwe (Holland) by processing the bottom half of the Tandem images 3512 (ERS2) and 23185 (ERS1). The interferogram was multilooked by 40x8, resulting in 367 lines and 610 pixels. There were about 3 fringes.

Baseline:
$B = 185m, \alpha = 3^\circ$
$B_\parallel = -62, B_\perp = -173$
$B_h = -184, B_v = -2$

The processing was done with a debugger version of the Doris software, so the cpu times are not really representative for the performance of Doris. table 32.1 shows some processing parameters.

Figure 32.2 and 32.3 show plots for these three methods. Figure 32.4 shows a comparison between schwabisch and ambiguity method.

It can be seen there is a trend between schwabisch and ambiguity. And there is an offset with rodriguez.

Schwabisch method is always higher then ambiguity, suggests error in computation of baseline parameters?

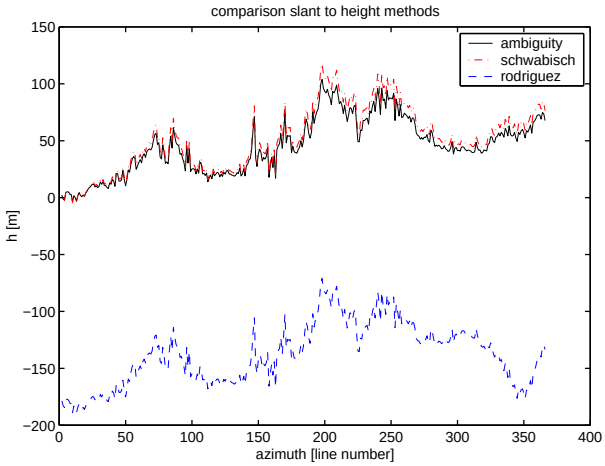

Figure 32.2: Comparison methods in azimuth direction for line 250.

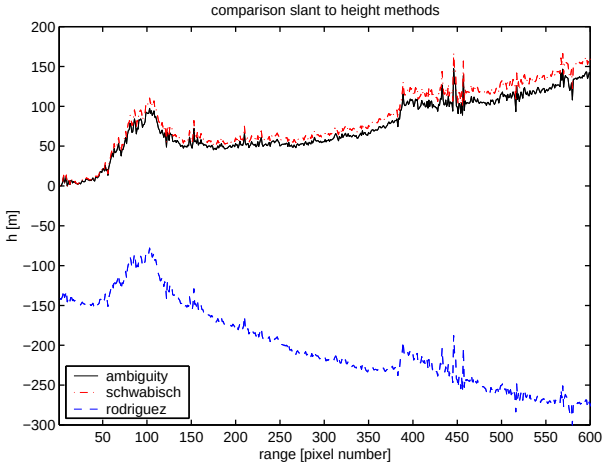

Figure 32.3: Comparison methods in range direction for pixel 110.

Table 32.1: Processing with the methods

Method	cpu	options	remarks
Ambiguity	60	-	geocoding as well
Schwabisch	12	1000 pnts, 1d=2, 2d=5	-
Rodriguez	4	-	-

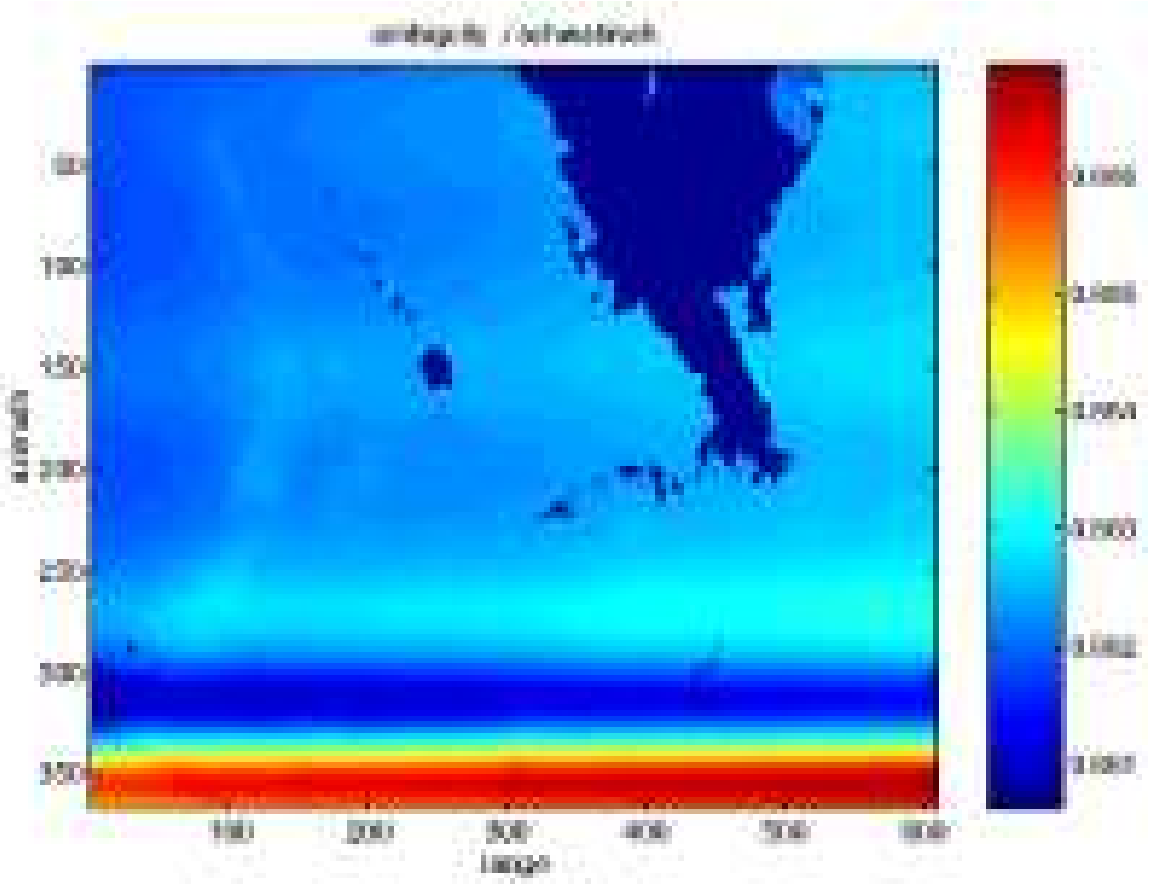

Figure 32.4: Comparison ambiguity, schwabisch methods for total image.

Some other tests also showed that the method Schwabisch, as implemented in Doris, seems to yield a (more or less) scaled version of the height of the ambiguity method. (The schwabisch being higher).

After rescaling (with a factor 0.86) of the heights obtained by schwabisch method to the level of the ambiguity method, the differences between both methods were only a few meters.

Chapter 33

GEOCODE

In this chapter the processing of step GEOCODE is described. In this step the radar coded heights are converted to geocoded coordinates (i.e., to a known reference system).

Input is the height file from the step SLANT2H. Output are two files containing the latitude (phi) and longitude (lambda) corresponding to the height file.

These files can be further processed with programs cpxfiddle, proj, and GMT to create a DEM in a regular grid in any projection desired (UTM for example). See the bin directory and the shell scripts there for examples how to do this. Likely, for each specific application, these scripts are best copied and adapted to your needs. The interpolated matrices from GMT are in grd format that can be handled by Matlab, etc.

33.1 Input Cards

GEO_OUT_PHI *geo_phi.raw*
Output file name for latitude.

GEO_OUT_LAM *geo_lam.raw*
Output file name for longitude.

Example input:

```
 c
 c
 comment   ___ GEOCODING ___
 c
GEO_OUT_LAM   Outdata/lam.raw
GEO_OUT_PHI   Outdata/phi.raw
```

If you want to obtain the latitude/longitude of the pixels in an interferogram that was created, but you do not have a DEM in radarcoordinates available, you will have to create one. This means that you will have to edit the **products result file** and create a SLANT2H section, see Chapter 32 for a description of this section in the **products result file**. The section will look something like:

```
*********************************************************************
*_Start_slant2h:
*********************************************************************
Method:              schwabisch
Data_output_file:                       Outdata/dummy_height.raw
Data_output_format:                     real4
First_line (w.r.t. original_master):    1001
Last_line (w.r.t. original_master):     2105
First_pixel (w.r.t. original_master):   501
Last_pixel (w.r.t. original_master):    700
Multilookfactor_azimuth_direction:      10
Multilookfactor_range_direction:        2
Ellipsoid (name,a,b):                   WGS84 6.37814e+06 6.35675e+06
*********************************************************************
* End_slant2h:_NORMAL
*********************************************************************
```

(Also set the pcf to 1 on top of the **products result file**.) If you have an external DEM you can compute the required file (Outdata/dummy_height.raw) using step COMPREFDEM. The dimensions and multilooking can be copied from the interferogram section.

If your area is flat, you may want to use a dummy file filled with zeros. You can create such a file with appropriate dimensions using Matlab for example. Alternatively, the much faster Unix way would be along these lines. First compute the height (number of lines) of the dummy file:

```
echo "(2105-1001+1)/10" | bc -l 110.5
```

then the width (number of pixels):

```
echo "(700-501+1)/2" | bc -l 100
```

(ie., the file should be 110 lines by 100 pixels of 4 byte). Now create the file using dd:

```
dd if=/dev/zero of=Outdata/dummy_height.raw count=110 bs=400
```

33.2 Output Description

The **process control flag** is switched at successful exit:

```
geocoding:              1
```

An example of the output in the **products result file**:

```
*********************************************************************
*_Start_geocode
*********************************************************************
Data_output_file_hei (slant2h):         Outdata/hei.ambi
Data_output_file_phi:                   Outdata/phi.raw
Data_output_file_lamda:                 Outdata/lambda.raw
Data_output_format:                     real4
First_line (w.r.t. original_master):    1001
Last_line (w.r.t. original_master):     2105
First_pixel (w.r.t. original_master):   501
Last_pixel (w.r.t. original_master):    700
Multilookfactor_azimuth_direction:      10
Multilookfactor_range_direction:        2
*********************************************************************
* End_geocode:_NORMAL
*********************************************************************
```

33.3 Implementation

Known are the heights of each pixel in the master (line,pixel) system. The point P(x,y,z) corresponding to a (line,pixel) is computed with the 3 equations (see Annex D) in such a way that it lies on an ellipsoid of height h above the refernce ellipsoid. When these coordinates are known, the equations of Bowring are used to transform them to an ellipsoid system (ϕ, λ, h). The semimajor axis is denoted by *a*, and the semiminor axis is denoted by *b*. The squared first eccentricity by:

$$e^2 = \frac{a^2 - b^2}{a^2} \tag{33.1}$$

The squared second eccentricity by

$$e'^2 = 1 - e^2 = \frac{a^2 - b^2}{b^2} \tag{33.2}$$

$$\begin{aligned} r &= \sqrt{x^2 + y^2} && (33.3)\\ \nu &= \arctan_2(z \cdot a), (r \cdot b)) && (33.4)\\ sin3 &= \sin^3 \nu && (33.5)\\ cos3 &= \cos^3 \nu && (33.6) \end{aligned}$$

$$\begin{aligned} \phi &= \arctan_2((z + e'^2 \cdot b \cdot sin3), (r - e^2 \cdot a \cdot cos3)) && (33.7)\\ \lambda &= \arctan_2(y, x) && (33.8)\\ N &= \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} && (33.9)\\ h &= \frac{r}{\cos \phi} - N && (33.10) \end{aligned}$$

Bibliography

[Arikan et al., 2008] Arikan, M., van Leijen, F., Guang, L., and Hanssen, R. (2008). Improved image alignment under the influence of elevation. In *Fifth International Workshop on ERS/Envisat SAR Interferometry, 'FRINGE07', Frascati, Italy, 26 Nov-30 Nov 2007*, page 4 pp.

[Bähr and Vögtle, 1991] Bähr, H. P. and Vögtle, T. (1991). *Digitale Bildverarbeitung: Anwendung in Photogrammetrie, Kartographie und Fernerkundung*. Wichmann Verlag, Karlsruhe.

[Curlander and McDonough, 1991] Curlander, J. C. and McDonough, R. N. (1991). *Synthetic aperture radar: systems and signal processing*. John Wiley & Sons, Inc, New York.

[Eineder, 2003] Eineder, M. (2003). Efficient simulation of SAR interferograms of large areas and of rugged terrain. *IEEE Transactions on Geoscience and Remote Sensing*, 41(6):1415–1427.

[Gatelli et al., 1994] Gatelli, F., Monti Guarnieri, A., Parizzi, F., Pasquali, P., Prati, C., and Rocca, F. (1994). The wavenumber shift in SAR interferometry. *IEEE Transactions on Geoscience and Remote Sensing*, 32(4):855–865.

[Geudtner, 1996] Geudtner, D. (1996). The interferometric processing of ERS-1 SAR data. Technical Report ESA-TT-1341, European Space Agency. Translation of DLR-FB 95-28.

[Geudtner and Schwäbisch, 1996] Geudtner, D. and Schwäbisch, M. (1996). An algorithm for precise reconstruction of InSAR imaging geometry: Application to "flat earth" phase removal, phase-to-height conversion, and geocoding of InSAR-derived DEMs. In *European Conference on Synthetic Aperture Radar, Königswinter, Germany, 26–28 March 1996*, Könogswinter, Germany.

[Ghiglia and Pritt, 1998] Ghiglia, D. C. and Pritt, M. D. (1998). *Two-dimensional phase unwrapping: theory, algorithms, and software*. John Wiley & Sons, Inc, New York.

[Goldstein and Werner, 1998] Goldstein, R. M. and Werner, C. L. (1998). Radar interferogram filtering for geophysical applications. *Geophysical Research Letters*, 25(21):4035–4038.

[Hanssen and Bamler, 1999] Hanssen, R. and Bamler, R. (1999). Evaluation of interpolation kernels for SAR interferometry. *IEEE Transactions on Geoscience and Remote Sensing*, 37(1):318–321.

[Nitti et al., 2008] Nitti, D. O., Hanssen, R. F., Refice, A., Bovenga, F., Milillo, G., and Nutricato, R. (2008). Evaluation of DEM-assisted SAR coregistration. In *SPIE Europe Remote Sensing, Proceedings 15–18 September 2008, Cardiff, United Kingdom*, pages 1–14.

[Rodriguez and Martin, 1992] Rodriguez, E. and Martin, J. M. (1992). Theory and design of interferometric synthetic aperture radars. *IEE Proceedings-F*, 139(2):147–159.

[Samson, 1996] Samson, J. (1996). Coregistration in SAR interferometry. Master's thesis, Faculty of Geodetic Engineering, Delft University of Technology.

[Scharroo and Visser, 1998] Scharroo, R. and Visser, P. (1998). Precise orbit determination and gravity field improvement for the ERS satellites. *Journal of Geophysical Research*, 103(C4):8113–8127.

[Schwäbisch, 1995] Schwäbisch, M. (1995). Die SAR-Interferometrie zur Erzeugung digitaler Geländemodelle. Forschungsbericht 95-25, Deutsche Forschungsanstalt für Luft- un Raumfahrt, Oberpfaffenhofen.

[Shewchuk, 1996] Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Lin, M. C. and Manocha, D., editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag. From the First ACM Workshop on Applied Computational Geometry.

[Shewchuk, 2002] Shewchuk, J. R. (2002). Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):21–74.

[Touzi et al., 1996] Touzi, R., Lopes, A., and Vachon, P. W. (1996). Estimation of the coherence function for interferometric SAR applications. In *European Conference on Synthetic Aperture Radar, Königswinter, Germany, 26–28 March 1996*, pages 241–244.

[Wessel and Smith, 1998] Wessel, P. and Smith, W. H. F. (1998). New, improved version of generic mapping tools released. *EOS Transactions, AGU*, 79(47):579.

[Zebker et al., 1994] Zebker, H. A., Rosen, P. A., Goldstein, R. M., Gabriel, A., and Werner, C. L. (1994). On the derivation of coseismic displacement fields using differential radar interferometry: The Landers earthquake. *Journal of Geophysical Research*, 99(B10):19617–19634.

Annex A

What's new?

A.1 Version 4.02

- The script *doris.process-reset.sh* is renamed to *doris.rmstep.sh* and is updated for Mac OS X.
- *plotcpm* is updated to plot offsets rather than the offset residuals. Another script is made available under the name *plotcpm.residues* to plot the offset residuals.

Bug fixes:

- Step **COMPREFDEM**: when multilooking applied during computation of reference DEM phase, the output file was smaller than the expected size.
- Step **COMPREFPHA**: fixed index overflow in sinus look up table.

A.2 Version 4.01

- Handling of file sizes larger than 4GB on both 32-bit and 64-bit platforms.
- Processing of TerraSAR-X data.
- Improvement of the master-slave overlap calculation. The new algorithm should prevent segmentation faults as sometimes experienced in the past.
- Improvement and speed-up of **COMPREFDEM** (see Chapter 26) based on a two-step approach. First, the complete DEM is radarcoded and written to a file. Second, the radarcoded DEM is interpolated to the master image geometry using Delaunay triangulation. The overlapping buffers, as in previous versions of Doris, are prevented.
- Simulation of master amplitude image based on DEM (**M_SIMAMP**), see Chapter 6.
- Estimation of master timing error based on correlation between master amplitude and simulated master amplitude (**M_TIMING**), see Chapter 7.
- Estimation of relative timing error between master and slave image (**RELTIMING**), see Chapter 18.
- DEM assisted coregistration (**DEMASSIST**), see Chapter 19. For optimal performance, apply after **M_SIMAMP**, **M_TIMING** and **RELTIMING**.
- Coherence estimation after DEM subtraction (default), see Chapter 28.
- Utility *construct_dem.sh* to download, merge and fill voids of SRTM data (see Section C.2.12).

- Utility *doris.process-reset.sh* to reset and clean up the processing entries in Doris result files allowing to easily re-run a step or multiple steps (see Section C.2.13).
- Option to output height-to-phase conversion factors (H2PH) based on flat earth (**SUBTRREFPHA**) or DEM (**COMPREFDEM**).
- During **COARSECORR** and **FINE** coregistration NaN (not-a-number) correlation values, which are mainly due to no data regions at the border of the scene, are eliminated automatically, but the whole list of windows are kept in the doris.log file.
- Coherence output is properly scaled between 0 to 1 range with new option **-r** of **cpxfiddle**, see Chapter C.2.3.

Annex B

Installation

In this annex the installation of Doris is described. To properly compile the Doris software, you might have to edit the Makefile. Set the compiler (CC), compiler flags (CFLAGS), library path (LFLAGS), and defines (comment DEF4 and DEF5 for VECLIB/LAPACK library usage). Use DEF7 (__X86PROCESSOR__) if you have a little endian machine.

We have written a simple script ”configure” to help generate a user defined Makefile, which is present in the Doris distribution, as well as a template Makefile that can be edited to your likings if the script fails.

We recommend compiling 2 versions of the Doris software. An optimal version for operational processing, and a more verbose debug version that is only to be used if the optimal version exits with an unexplained error. (Then repeat processing with debug version, track down routine, etc.)

Compilation of these two versions is best done by running ”make” two times, first with CFLAGS = CFLAGSOPT, then (after make clean) with CFLAGS = CFLAGSDEBUG. This is clearly described in the Makefile and in the Makefile generator.

We have successfully compiled Doris with GNU g++ 2.x, 3.x, 4.x, Sun compilers, Intel compilers and so on.

If you have problems installing Doris, you can sent your questions to the email list of doris users. To join this list, follow the directions at our internet site (http://enterprise.lr.tudelft.nl/doris/). Please don’t forget to specify platform, compiler, versions etc.

B.1 Installation of Doris

B.1.1 Installation of the Doris core

After downloading the gzipped, tarred archive of the Doris software v4.02the installation is best done with a Makefile. I assume you are familiar with ’make’ to compile code. If you are not, find someone who is.

1. Create a directory for Doris, e.g., mkdir /opt/doris_v4.02
   cd /opt/Doris_v4.02
2. Download the archived Doris software via the download area of our webpages at
   http://enterprise.lr.tudelft.nl/doris/: **doris_v4.02.tar.gz**
3. Expand compressed files: gzip -d doris_v4.02.tar.gz
   This leaves a file **doris_v4.02.tar**
4. Extract the files from the archive:
   tar -xvf doris_v4.02.tar

Now sub directories are created, **bin**, **src**, **SARtools**, **ENVISAT_TOOLS**. The doris source files are in src and bin. The other two are utilities that need to be compiled separately.

Now we are ready to compile Doris. cd to the src directory, and read the README file for the latest information.

1. Create a Makefile by running the script: **configure**
   in the **src** directory. ("csh configure" or "chmod 755 configure" if it is not executable.) This should limit the editing in the Makefile. Follow the directions on the screen.
2. Compile Doris and install the executables:
   - (in directory Source_new; inspect/edit the Makefile)
   - make (this compiles the code)
   - make test (this should give the version number of Doris)
   - make install (uses /usr/local/bin/ by default. Also the bin utilities are installed.)
3. Make sure the Installation directory is in your path. For (t)csh users, it should be in your .(t)cshrc file (startup file). Add it with a (csh) command like:

   ```
   set path = ( /usr/local/bin $path )
   ```

B.1.2 Installation of the SARtools

We also need to compile the SARtools and ENVISAT_TOOLS programs. There are Makefiles in the sub-directories provided, with default installation directories. If you selected another installation directory than /usr/local/bin, please change that in the two Makefiles. What you have to do to install these utilities is:

1. cd SARtools
2. (review the Makefile)
3. make -n (check what happens)
4. make (compile software)
5. make -n install (check if this is what you want)
6. make install

B.1.3 Installation of the ENVISAT_tools

And for the ENVISAT_TOOLS

1. cd ENVISAT_TOOLS
2. (review the Makefile)
3. make -n (check what happens)
4. make (compile software)
5. make -n install (check if this is what you want)
6. make install

We did not automate this because of the complexity, while simply editing two Makefiles should not be a big problem.

B.1.4 Installation of the TERRASAR-X reader

For the use of Terrasar-X data, the following additional packages are required on your system (note the minimum version numbers):

- gdal (version $\geq$ 1.44). See `http://gdal.org` for more information.
- python (version $\geq$ 2.2).
- libxml2 (version $\geq$ 2.7.2).
- python-lxml (version $\geq$ 2.0).
- libxslt (version $\geq$ 1.1.15).

In case your system does not meet these requirements and you cannot update, you can try an alternative script. In that case, the requirements are

- gdal (version $\geq$ 1.44). See `http://gdal.org` for more information.
- python (version $\geq$ 2.2).
- libxml2 (version $\geq$ 2.6.30).
- python-lxml (version $\geq$ 1.3.3-1).
- libxslt not required.

To use the alternative script, go to your Doris bin directory and do:

```
cp tsx_dump_header2doris_no_xpath.py tsx_dump_header2doris.py
```

to overwrite the original script.

B.1.5 Starting Doris ...

Now (after a rehash) we can run the Doris software and start InSAR processing! The **run** script in the **bin** directory can be customized by setting the environment variables **EDITOR** and **PAGER**.

Note that it is *highly adviced* to install the **utilities**, see annex C, that can be found in the download area, and **GMT** (visualization).

To use the Delft precise orbits it is convenient to have **getorb** on your system, but one could also use the online version from http://www.deos.tudelft.nl/ers/precorbs/ .

B.1.6 Installation of utility scripts

In the Doris archive there are a number of utilities included which are required for optimal processing. (They can be called from within Doris.)

The utilities in the bin directory are explained in section B.6. If these files are in your path and they can be executed then they are installed ok. Doris writes the calling syntax of these utilities to standard out (as INFO), so you can repeat the commands. For example (assuming the run script is used):

```
grep plotoffsets Outinfo/out.*
grep plotcpm Outinfo/out.*
```

yields for example

```
.Outinfo/out.input.fine_cpm.4926:INFO:     plotcpm CPM_Data 1 5000 1 1000&
```

This command can be repeated from the prompt.

B.2 Additional programs

To obtain a full version of Doris, the **getorb** (precise orbits), **GMT** (mapping tool, helper for fine co-registration) and **gv or ghostview** (to display postscript files in csh-script plotscript) should be installed on your system. These programs can be obtained freely, but are not included in the Doris distribution.

At our homepage you can find out more on how to obtain these packages. (http://enterprise.lr.tudelft.nl/doris/).

B.3 Running the Doris software

You can run the Doris software by making an **input file** as described in this user's manual. For full functionality, make sure that getorb/ghostview/gmt are in your path (add them in your `$home/.cshrc` file or in the `$home/.login` file).

The command line options for Doris are:

- doris -v
  Return version number.
- doris -h [search pattern]
  Return help for "search pattern". (Calls the shell script helpdoris.)
- doris -q
  Return random quote. (This option can be used to add random quotes to your mail. Make an alias for e.g. elm or pine (mailprograms): "alias elm 'doris -q ¿ /.signature; elm'" Then, the next time elm is called, it first creates a .signature file in your home directory, which is appended to your mail message.)
- doris -c
  Return copyright notice.
- doris inputfile
  Run doris with the input specified in "inputfile" (defaults to "inputoptionsfile").

For convenience we have developed a simple csh-script (named "run" in the BIN directory) that can generate a template input for you and that works as a shell for the actual processing.

You can run the Doris software by (for example, I assume you know the vi editor):

1. Make a directory: mkdir /data/kampes/Testdoris
   Go to that directory: cd /data/kampes/Testdoris
2. Copy the run file for editing: cp /home/Doris/BIN/run .
   Edit the run file, for bin directories, and remarks: vi run
   Take a look at the help: run -h Generate input templates: run -g
3. Edit the generated **input file**: run -e1
   Run the first step: run -s1

4. View output stout: run -v1
   View output **result file**: run -r1
   View output logfile: run -r4

5. Process next step: run -e2; run -s2; run -v2; etc.

B.4 Viewing the results of Doris

Annex C describes a number of tools to visualize the output of Doris. The parameter files, log file, etc. can be viewed with any standard editor. For this, the **run** script utility can also be used, customized by setting the PAGER and EDITOR environment variables.

The data output are binary files, which can be visualized with standard software on your system, e.g., Khoros, Matlab (tip: try spinmap), etc.

The utility **cpxfiddle** (c++ in SARtools archive) can, amongst other things, generate SUNraster files of the phase, magnitude, and magnitude+phase with a colormap of your liking (use, e.g., ”xv” or ”display” to view/print.) This seems to be one of the fastest ways to see the results. See also the PREVIEW card.

The utility **cpx2ps** (csh script) can generate postscript code from complex files for the magnitude or phase. (use gv or ghostview to view/print.) This utility uses the GMT package. (Tip: generate 2 m postscript files, with -Z option, and view this enlarged interferogram with gv.)

If someone want to develop a Motif/Lesstif X windows application that would be very welcome. We have experimented with these things, but no time to implement it in a robust way.

A tip might be to use X utilities like xmag or xlupe to zoom in on the visualized files.

B.5 Trouble shooting

B.5.1 General problems

In Tables B.1 and B.2 a number of possible problems and their solutions is given.

Table B.1: Troubleshooting #1

Problem	Solution
Function FFT not found while linking	You have no Veclib library. Change define statements in Makefile. (Run the install script that generates the Makefile again.) Consider adding a (library) routine for FFT to doris, since the internal one is not optimized speedwise.
Compilation fails due to int16 problem with var arg in ioroutines.c	Change code in ioroutines.c, routine checkrequest, from int16 to int. (e.g. int16 N = va_arg(arglist, int16) $-$ $>$ int N = va_arg(arglist, int); This happened with redhat 7.0, but not sure if Doris still functions completely correct; It should have no influence on the computations, only on user friendlyness, i.e., warnings if Doris thinks certain steps should not be run.)
Compilation fails because compiler cannot find include files	Change include files in source (e.g. $\#include < cctype >$ to $\#include < ctype.h >$). Particularly older compilers do not know of the new standard (e.g., cctype).
Compilation fails due to strptime (Redhat 7.0?)	Remove comment before DEF8 in the Makefile. (i.e.: DEF8 = -D_NO_STRPTIME). (Remove object files with command "make clean" not required). Compile again with command "make".
Run script doesn't work at all	Check first line of run script. There the interpreter (an executable program) is specified, for example !/bin/ksh. Does this program exist? (test with command: "which ksh"). For redhat 7.0, who doesn't include it in the distribution for some strange reason, search the internet for a public domain ksh (korn shell). (search at "www.google.com" on "+pdksh +download") Is the location correct? (make a symbolic link as root)

Table B.2: Troubleshooting #2

Problem	Solution
Run script doesn't work properly	Path not set to bindir (try if command "doris -v" returns version number) or GMT is not installed. Or directory not writable (chmod).
Run script: editing option doesn't work properly/not to my liking	Use the environment variable EDITOR to specify your preferred editor, for csh, include in your startup file .cshrc for example a line "setenv EDITOR vi"
Run script: viewing option doesn't work properly/not to my liking	Use the environment variable PAGER to specify your preferred viewer, for csh, include in your startup file .cshrc for example a line "setenv EDITOR vi"
Doris crashes at getting the Delft orbits	Is getorb installed? Check location of orbdir (Doris input card). Obtain the system command Doris parses from the stdout INFO, repeat it from the command line.
Doris gets zero correlation in FINE coregistration.	Is path to files correct (in master.res, slave.res)? Compile a debug version of doris and run that to find out what goes wrong.
Doris cannot generate postscript while coregistration.	Is GMT installed? Check script from prompt by using command that is written as INFO to stdout. Consider using NOPLOT options in Doris input.
Script plotoffsets or plotcpm does not work.	Read the help by typing plotoffsets at the prompt. View the script in an editor, try to detect the error. adding -x to the first line of the script should echo all commands before execution. Run it without the background option. What awk are you using? We use gawk actually, but it should be standard POSIX. What is the correct syntax for tail? ("-n +6" or "-n+6" or "+6", the tail commands have been removed from the new version)?
Doris leaves temporary files in the working directory.	This should not happen, but sometimes these files are not removed. If Doris exits normally, these files can be savely removed. If Doris exits with an error, sometimes these files can be used to repair the **result files**.
What is this file CPM_data after FINE coregistration?	This ascii file is used to generate plots with GMT.
Doris crashes on reading input.	Try to find out with the debug version of Doris where it exactly goes wrong. Some cards with optional ON OFF options need one of these specified on some systems.

B.5.2 Matrix class troubles

Some users reported having problems compiling a template class which was implemented in different files. I hope that by explaining my problems I might help someone else.

For the aCC compiler the flag +inst_implicit_include had to be used. This flag include the file matrixklasse.cc (note: .cc) automatically.

Furthermore, for making an archive library, the member functions had to be instantiated explicitly, see file matlib.c for how I did that. The friend functions also had to be instantiated, also see the Makefile and matlib.c.

We previously ran into some problems with the gnu g++ compiler version 2.95.2 and were not able to compile Doris (explicit instantiation mechanism/parsing of friend template functions, bug in this version?). We solved this problem by putting all definitions of the matrix class in one file and compiling without first creating a matrix library. Doris v2.4 and higher should therefor be more compiler/platform independent. For developers this is not quite so comfortable, because the code is less transparent this way.

If compilation as described above gives problems, try the following:

1. Try **make -n processor.o** (only echos command to the screen) and paste this from the prompt to give you more direct control.
2. Do not use Veclib and Lapack libraries, even if you have them. To exclude them, define DEF4 and DEF5 in the Makefile (uncomment the defines). Now you will be making use of the internal routines, which are based on numerical recipes routines.
3. Set verbose flags for compilation (-v likely, add to CFLAGS).
4. Try to compile object code for a individual source files, e.g., **make processor.o** and if successful, try others. **make swobjs** does all.
5. If all .o files are compiled correctly, use **make** or **make doris** to link them.
6. Try another compiler (first **make clean**).

B.5.3 Some notes on installation on SGI

This message was posted to the doris_users email list. It may be of help if you are installing Doris on a SGI platform. Thanks to Kamini Kanta Mohanty.

```
From: Kamini Kanta Mohanty <mohantykk@yahoo.com>
Subject: My Experience on DORIS
To: Bert Kampes <kampes@geo.tudelft.nl>
Cc: doris_users@tudelft.nl, kkm_10@hotmail.com

+-------------------------------------------------------------+
| Doris Listserver                                            |
| (Delft Object-oriented Radar Interferometric software)      |
|                                                             |
+-------------------------------------------------------------+

From: K.K. Mohanty
Marine and Water Resources Division
Space Applications Centre (ISRO)
Ahmedabad - 380 053, INDIA
mohantykk@yahoo.com
 7 July, 2000
To

Dear Doris Users,

 At the outset, I would like to thank Mr. Bert Kampes,
DEOS, Delft University to make DORIS openly available
```

for download. I have downloaded DORIS 2.3 software sometime in the middle of May 2000. Subsequently, I have installed the same in SGI (Silicon Graphics) Octane w/s with IRIX 6.4 o/s. I have executed many steps in the s/w (**not** all). I would like to share my experience of installing s/w in SGI machine. Also, got some queries. May be a few suggestions, which can be taken care of in future release.

I am **new** to interferometry. I am based at Space Application Centre (ISRO), Ahmedabad, INDIA. I am also an ITC, Netherlands alumni.

Experience During Installation:
---

1) complex type in SGI is a **class** (**not template**) supporting only **double** type. I had to separately bring in the complext.h from GNU. It works fine.

2) Equivalents of all other includes are available, but .h extension has to be added (**for** example, <iostream> has to be replaced by <iostream.h>)

3) The ios::binary mode in file open is **not** required in IRIX 6.4. I understand it's true for many unix variants. I simply removed it, and then it works.

4) In all declarations such as, template matrix<TYPE> operator * TYPE2 (const matrix<TYPE> &A, const matrix<TYPE> &B); in matlib.c the keyword template has to be replaced by keyword class. This is also true for declaration for member functions such as template matrix<TYPE> correlate TYPE2 (const matrix<TYPE> &A, const matrix<TYPE> &B);

5) During final linking for making doris executable, an undefined symbol void matrix<complex<float>>::conj() was complained by the linker, even though it was being generated by conditional comipling for creation of matrix library in matrixbaseclass.overloaded. This has mostly to do with order of linking the libraries. I avoided this by specializing the corresponding function void matrix<TYPE>::conj() as void matrix<complex<float>>::conj() in matrixbaseclass.ovrloaded.

6) I get a lot of warning stating multiply defined:(malloc_alloc::oom_malloc() for different primitive types. They can be ignored.

7) A final error in ioroutine.c while compiling in debug mode for line if(compl4(1.1) != complr4 (1.1, 0) stating more than one != matches was reported. Since, it was only for debugging mode, I avoided this by commenting this line. This may have to do with my implementation of complext.h from GNU.

Experience Running Doris

[...]

Mohanty, KK

[...]

B.5.4 Some notes on installation on Linux X86

Since July 2000 Doris can be installed to Linux X86 systems. The code needed some changes. Please refer to the general problems section as well.

- Byte order on X86 systems. Use the functions htonl and htons to read the record length and data of the SLC volume, leader, and data file.
- File io.
  - (gcc compiler only?) ios::ate does not act as expected. It should open an **input file** stream at the end, but it does not. ios::app (ofstream class) does work, as does a file.seekg(0,ios::end);
  - ios::in (or out) has to be set, even for statements like ifstream ifile(ios::nocreate); The default and logical, that a file is an input stream if it is declared ifstream, is not true for gcc compiler.
- strcmp. The statement strcmp(word,'\0') gives a memory fault if tested.

B.5.5 Some notes on installation on Window running Cygwin

Bert Kampes reported installing Doris on Windows NT and XP running Cygwin without trouble beginning of 2002. The run/installation scripts needed some small changes. Here is the summary. It is assumed that a full version of Cygwin installed, including tcsh, developing tools.

- There is no ksh and csh for Cygwin in standard setup? Therefor change the first line of the configure script to tcsh, and use the new versions of the helpdoris and run script where a sh implementation is used (instead of ksh). I linked csh to tcsh in the bin directory of my cygwin installation, which also works fine, and prevents changing the scripts, i.e., *ln -s /bin/tcsh /bin/csh*.
- Install GMT tools.
- Install XFree86 and ghostview, xv, etc.
- i didn't install getorb, and that may partly use a fortran compiler. (If someone has done this, please let me know.)
- For convenience make a symbolic link to your cdrom drive, e.g., *ln -s d: /cdrom*. You can then refer to SLC files on cdrom in the Doris input with /cdrom/SCENE1/...

B.6 List of files in archive

The following directories are created after "tar -xvf Dorisv1.0.tar":

- Bin csh-scripts, helper programs.
- Source_new Source code Doris.

For example the following files are in the archive Dorisv2.5.tar (11 July 2000):

```
r--r--r--  413/22    5767  Jul   7 17:47 2000  Source_new/\makefile
r--r-----  413/22    8998  Jul   7 17:47 2000  Source_new/constants.h
r--r-----  413/22   19441  Jul   7 17:47 2000  Source_new/conversion.c
r--r-----  413/22    5641  Jul   7 17:47 2000  Source_new/conversion.h
r--r-----  413/22  127641  Jul   7 17:47 2000  Source_new/coregistration.c
r--r-----  413/22    5020  Jul   7 17:47 2000  Source_new/coregistration.h
r--r-----  413/22   20883  Jul   7 17:47 2000  Source_new/filtering.c
```

```
r--r------ 413/22    1410 Jul  7 17:47 2000 Source_new/filtering.h
r--r------ 413/22   60217 Jul  7 17:47 2000 Source_new/geocode.c
r--r------ 413/22    3549 Jul  7 17:47 2000 Source_new/geocode.h
r--r------ 413/22  101161 Jul  7 17:47 2000 Source_new/ioroutines.c
r--r------ 413/22    5289 Jul  7 17:47 2000 Source_new/ioroutines.h
r--r--r-- 413/22   29980 Jul  7 17:47 2000 Source_new/matdoc.txt
r--r--r-- 413/22   70878 Jul  7 17:47 2000 Source_new/matrixbk.cc
r--r--r-- 413/22   57993 Jul  7 17:47 2000 Source_new/matrixspecs.c
r--r------ 413/22   51646 Jul  7 17:47 2000 Source_new/processor.c
r--r------ 413/22   56881 Jul  7 17:47 2000 Source_new/products.c
r--r------ 413/22    2587 Jul  7 17:47 2000 Source_new/products.h
r--r------ 413/22  130910 Jul  7 17:47 2000 Source_new/readinput.c
r--r------ 413/22   15893 Jul  7 17:47 2000 Source_new/readinput.h
r--r------ 413/22   42007 Jul  7 17:47 2000 Source_new/referencephase.c
r--r------ 413/22    3162 Jul  7 17:47 2000 Source_new/referencephase.h
r--r------ 413/22    1106 Jul  7 17:47 2000 Source_new/refsystems.h
r--r------ 413/22   72665 Jul  7 17:47 2000 Source_new/step1routines.c
r--r------ 413/22    1121 Jul  7 17:47 2000 Source_new/step1routines.h
r--r------ 413/22   11938 Jul  7 17:47 2000 Source_new/unwrap.c
r--r------ 413/22     844 Jul  7 17:47 2000 Source_new/unwrap.h
r--r------ 413/22   66820 Jul  7 17:47 2000 Source_new/utilities.c
r--r------ 413/22    9630 Jul  7 17:47 2000 Source_new/utilities.h
r-xr-x--- 413/22   21633 Jul  7 17:47 2000 Bin/helpdoris
r-xr-xr-x 413/22   13534 Jul  7 17:47 2000 Bin/plotcpm
r-xr-xr-x 413/22   13093 Jul  7 17:47 2000 Bin/plotoffsets
rwxr-xr-x 413/22   29621 Jul  7 17:47 2000 Bin/run
r-xr-x--- 413/22     548 Jul  7 17:47 2000 Bin/viewanddel
```

You should add the Bin directory to your path. The utilities in the Bin directory are there for your convenience and you may edit them the way you prefer. Possibly you need to make these files executable by the command:

```
chmod 755 Bin/*
```

```
Bin/run          Generate input and shell for running Doris.
Bin/helpdoris    Summary of input keywords. print with helpdoris -p.
Bin/plotcpm      Step coregpm: GMT show estimated error offset-model, histograms, etc.
Bin/plotoffsets  Step fine: GMT to show offset vectors, thresholded on correlation.
Bin/viewanddel   Step coregpm: background call to gv, thereafter delete (dummy) file.
```

B.7 List of routines + description

The information in this section may be out of date.

The list is generated with the ctags command

```
~/.bin/ctags -u -x *.[ch]* | grep -v matrixspec |\
grep -v matrixbk | grep func | cut -c1-20,39-600
```

```
0   sqr                 constants.h     inline int16 sqr(int16 x) {return(x*x);}
1   sqr                 constants.h     inline int32 sqr(int32 x) {return(x*x);}
2   sqr                 constants.h     inline uint sqr(uint x) {return(x*x);}
3   sqr                 constants.h     inline real4 sqr(real4 x) {return(x*x);}
4   sqr                 constants.h     inline real8 sqr(real8 x) {return(x*x);}
5   in                  constants.h     inline real8 in(cn P) const // scalar product: r=P.in(Q);
6   out                 constants.h     inline cn out(cn P) const // cross product cn r=P.out(Q);
7   dist                constants.h     inline real8 dist(cn P) const // distance: d=P.dist(Q);
8   min                 constants.h     inline cn min(cn P) const // cn r=P.min(Q);
9   norm2               constants.h     inline real8 norm2() const // n=P.norm2();
10  norm                constants.h     inline real8 norm() const // n=P.norm();
11  normalize           constants.h     inline cn normalize() const // cn R=P.normalize();
12  angle               constants.h     inline real8 angle(cn A) const // angle=A.angle(B); //0,pi;
```

```
13  ecc1st_sqr           constants.h     inline void ecc1st_sqr() // first ecc.
14  ecc2nd_sqr           constants.h     inline void ecc2nd_sqr() // second ecc.
15  disp                 constants.h     inline void disp() const // show content
16  lines                constants.h     inline uint lines() const // return number of lines
17  pixels               constants.h     inline uint pixels() const // return number of pixels
18  operator =           constants.h     window& operator = (window X)
19  operator ==          constants.h     bool operator == (window X) const
20  operator !=          constants.h     bool operator != (window X) const
21  pol2xyz              conversion.c    void pol2xyz(
22  xyz2pol              conversion.c    void xyz2pol(
23  xyz2ell              conversion.c    void xyz2ell(
24  xyz2ell              conversion.c    void xyz2ell(
25  ell2xyz              conversion.c    void ell2xyz(
26  deg2rad              conversion.h    inline real8 deg2rad(real8 x) {return x * PI / 180.;}
27  deg2rad              conversion.h    inline real4 deg2rad(real4 x) {return x * PI / 180.;}
28  rad2deg              conversion.h    inline real8 rad2deg(real8 x) {return x * 180. / PI;}
29  rad2deg              conversion.h    inline real4 rad2deg(real4 x) {return x * 180. / PI;}
30  line2ta              conversion.h    inline real8 line2ta(real8 line, real8 ta1, real8 prf)
31  pix2tr               conversion.h    inline real8 pix2tr(real8 pixel, real8 tr1, real8 rangesamplingratex2)
32  pix2range            conversion.h    inline real8 pix2range(real8 pixel, real8 tr1, real8 rangesamplingratex2)
33  ta2line              conversion.h    inline real8 ta2line(real8 azitime, real8 ta1, real8 prf)
34  tr2pix               conversion.h    inline real8 tr2pix(real8 rangetime, real8 tr1, real8 rangesamplingratex2)
35  cr4toci2             conversion.h    inline compli16 cr4toci2(complr4 x)
36  coarseporbit         coregistration.c void coarseporbit(
37  coarsecorrel         coregistration.c void coarsecorrel(
38  coarsecorrelfft      coregistration.c void coarsecorrelfft(
39  corrfft              coregistration.c real4 corrfft(
40  distributepoints     coregistration.c matrix<uint> distributepoints(
41  getoffset            coregistration.c void getoffset(
42  finecoreg            coregistration.c void finecoreg(
43  coherencefft         coregistration.c real4 coherencefft(
44  coherencespace       coregistration.c real4 coherencespace(
45  coregpm              coregistration.c void coregpm(
46  getofffile           coregistration.c matrix<real4> getofffile(
47  cc4                  coregistration.c matrix<real4> cc4(
48  cc6                  coregistration.c matrix<real4> cc6(
49  ts6                  coregistration.c matrix<real4> ts6(
50  ts8                  coregistration.c matrix<real4> ts8(
51  ts16                 coregistration.c matrix<real4> ts16(
52  rect                 coregistration.c matrix<real4> rect(
53  tri                  coregistration.c matrix<real4> tri(
54  resample             coregistration.c void resample(
55  rangefilter          filtering.c     void rangefilter(
56  rfilterblock         filtering.c     void rfilterblock(
57  phasefilter          filtering.c     void phasefilter(
58  goldstein            filtering.c     matrix<complr4> goldstein(
59  smooth               filtering.c     matrix<real4> smooth(
60  smooth               filtering.c     matrix<real4> smooth(
61  spatialphasefilt     filtering.c     void spatialphasefilt(
62  convbuffer           filtering.c     matrix<complr4> convbuffer(
63  phasefilterspectral  filtering.c     void phasefilterspectral(
64  spectralfilt         filtering.c     matrix<complr4> spectralfilt(
65  azimuthfilter        filtering.c     void azimuthfilter(
66  blockazifilt         filtering.c     matrix<complr4> blockazifilt(
67  slant2hschwabisch    geocode.c       void slant2hschwabisch(
68  slant2hambiguity     geocode.c       void slant2hambiguity(
69  slant2hrodriguez     geocode.c       void slant2hrodriguez(
70  geocode              geocode.c       void geocode(
71  printcpu             ioroutines.c    void printcpu(
72  inittest             ioroutines.c    void inittest()
73  doinitwrite          ioroutines.c    bool doinitwrite(
74  initwrite            ioroutines.c    void initwrite(
75  updatefile           ioroutines.c    void updatefile(
76  getanswer            ioroutines.c    void getanswer(
77  readres              ioroutines.c    bool readres(
78  updateprocesscontrol ioroutines.c    void updateprocesscontrol(
79  checkprocessing      ioroutines.c    void checkprocessing(
80  checkrequest         ioroutines.c    void checkrequest(
81  fillcheckprocess     ioroutines.c    void fillcheckprocess(
82  fillprocessed        ioroutines.c    void fillprocessed(
83  filelines            ioroutines.c    int32 filelines(
```

```
84  existed              ioroutines.c   bool existed(
85  removedatleader      ioroutines.c   void removedatleader(
86  filesize             ioroutines.c   inline uint filesize(
87  getoverlap           ioroutines.c   window getoverlap(
88  getoverlap           ioroutines.c   window getoverlap(
89  readcoeff            ioroutines.c   matrix<real8> readcoeff(
90  fillproductinfo      ioroutines.c   void fillproductinfo(
91  assert               ioroutines.c   void assert(
92  assert               ioroutines.c   void assert(
93  tolower              ioroutines.c   void tolower(char *s)
94  toupper              ioroutines.c   void toupper(char *s)
95  printWARNING         ioroutines.h   inline void printWARNING()
96  ERROR                ioroutines.h   inline void ERROR(char ch[ONE27])
97  ERROR                ioroutines.h   inline void ERROR(const char* file, int32 line, char ch[ONE27])
98  WARNING              ioroutines.h   inline void WARNING(char ch[ONE27])
99  PROGRESS             ioroutines.h   inline void PROGRESS(char ch[ONE27])
100 INFO                 ioroutines.h   inline void INFO(char ch[ONE27])
101 DEBUG                ioroutines.h   inline void DEBUG(char ch[ONE27])
102 DEBUG                ioroutines.h   inline void DEBUG(const char* file, int32 line, char ch[ONE27])
103 initialize           orbitbk.cc     void orbit::initialize(const char* file)
104 computecoefficients  orbitbk.cc     void orbit::computecoefficients()
105 getklokhi            orbitbk.cc     void orbit::getklokhi(real8 t)
106 getxyz               orbitbk.cc     cn orbit::getxyz(
107 getxyzdot            orbitbk.cc     cn orbit::getxyzdot(
108 getxyzddot           orbitbk.cc     cn orbit::getxyzddot(
109 lp2xyz               orbitbk.cc     int32 lp2xyz(
110 xyz2orb              orbitbk.cc     int32 xyz2orb(
111 xyz2t                orbitbk.cc     int32 xyz2t(
112 xyz2lp               orbitbk.cc     int32 xyz2lp(
113 ell2lp               orbitbk.cc     int32 ell2lp(
114 lp2ell               orbitbk.cc     int32 lp2ell(
115 compbaseline         orbitbk.cc     void compbaseline(
116 dumporbit            orbitbk.cc     void orbit::dumporbit(
117 splineinterpol       orbitbk.cc     matrix<real8> splineinterpol(
118 showdata             orbitbk.cc     void orbit::showdata()
119 main                 orbitbk.cc     int32 main()
120 eq1_doppler          orbitbk.h      inline real8 eq1_doppler(cn velocity, cn dsat_P)
121 eq2_range            orbitbk.h      inline real8 eq2_range(cn dsat_P, real8 rangetime)
122 eq3_ellipsoid        orbitbk.h      inline real8 eq3_ellipsoid(cn P, real8 semimajora, real8 semiminorb)
123 eq1_doppler_dt       orbitbk.h      inline real8 eq1_doppler_dt(cn dsat_P, cn velocity, cn accerelation)
124 shownumberofpoints   orbitbk.h      int32 shownumberofpoints() {return numberofpoints;}
125 main                 processor.c    int32 main(
126 handleinput          processor.c    void handleinput(int argc, char* argv[], input_gen &input_general)
127 usage                processor.c    void usage(char *programname)
128 fillproductinfo      productinfo.cc void productinfo::fillproductinfo(
129 readphase            productinfo.cc matrix<real4> productinfo::readphase(
130 productinfo          productinfo.h  productinfo() {multilookL=1; multilookP=1;} // rest ==0
131 showdata             productinfo.h  inline void showdata() const // show content
132 compinterfero        products.c     void compinterfero(
133 compcoherence        products.c     void compcoherence(
134 subtrrefpha          products.c     void subtrrefpha(
135 subtrrefpha          products.c     void subtrrefpha(
136 subtrrefdem          products.c     void subtrrefdem(
137 dinsar               products.c     void dinsar(
138 writearg             readinput.c    void writearg(const Type argument)
139 readinput            readinput.c    void readinput(
140 checkgeneral         readinput.c    void checkgeneral(
141 checkreadfiles       readinput.c    void checkreadfiles(
142 checkcrop            readinput.c    void checkcrop(
143 checkporbits         readinput.c    void checkporbits(
144 checkslant2h         readinput.c    void checkslant2h(
145 checkunwrap          readinput.c    void checkunwrap(
146 checkgeocode         readinput.c    void checkgeocode(
147 checkcoarsecorr      readinput.c    void checkcoarsecorr(
148 checkfine            readinput.c    void checkfine(
149 checkcoregpm         readinput.c    void checkcoregpm(
150 checkcomprefpha      readinput.c    void checkcomprefpha(
151 checksubtrrefpha     readinput.c    void checksubtrrefpha(
152 checkresample        readinput.c    void checkresample(
153 checkinterfero       readinput.c    void checkinterfero(
154 checkcoherence       readinput.c    void checkcoherence(
```

```
checkcomprefdem      readinput.c     void checkcomprefdem(
checksubtrrefdem     readinput.c     void checksubtrrefdem(
checkfiltrange       readinput.c     void checkfiltrange(
checkdinsar          readinput.c     void checkdinsar(
checkfiltphase       readinput.c     void checkfiltphase(
checkfiltazi         readinput.c     void checkfiltazi(
setunspecified       readinput.h     inline void setunspecified(char *s)
specified            readinput.h     inline bool specified(const char *s)
flatearth            referencephase.c void flatearth(
radarcodedem         referencephase.c void radarcodedem(
fillslcimage         slcimage.cc     void slcimage::fillslcimage(const char* file)
updateslcimage       slcimage.cc     void slcimage::updateslcimage(
readdata             slcimage.cc     matrix<complr4> slcimage::readdata(
showdata             slcimage.h      inline void showdata() const
readvolume           step1routines.c void readvolume(
readleader           step1routines.c void readleader(
readnull             step1routines.c void readnull(
readdat              step1routines.c void readdat(
writeslc             step1routines.c void writeslc(
unwraptreeframon     unwrap.c        void unwraptreeframon(
getorb               utilities.c     void getorb(
convertgetorbout     utilities.c     void convertgetorbout(
solve33              utilities.c     void solve33(
solve22              utilities.c     matrix<real8> solve22(
nextpow2             utilities.c     uint nextpow2(
polyval              utilities.c     real8 polyval(
polyval              utilities.c     real8 polyval(
polyval              utilities.c     matrix<real4> polyval(
polyval1d            utilities.c     real8 polyval1d(
normalize            utilities.c     void normalize(
normalize            utilities.c     void normalize(
BBparBperp           utilities.c     void BBparBperp(real8 &B, real8 &Bpar, real8 &Bperp,
BBhBv                utilities.c     void BBhBv(
Btemp                utilities.c     int32 Btemp(
BalphaBhBvBparBperpT10  utilities.c    void BalphaBhBvBparBperpTheta(
iseven               utilities.h     inline bool iseven(int16 w) {return (w+1)%2;}
iseven               utilities.h     inline bool iseven(int32 w) {return (w+1)%2;}
iseven               utilities.h     inline bool iseven(uint w) {return (w+1)%2;}
isodd                utilities.h     inline bool isodd (int16 w) {return w%2;}
isodd                utilities.h     inline bool isodd (int32 w) {return w%2;}
isodd                utilities.h     inline bool isodd (uint w) {return w%2;}
ispower2             utilities.h     inline bool ispower2 (uint w)
Ncoeffs              utilities.h     inline int32 Ncoeffs(int32 degree)
degree               utilities.h     inline int32 degree(int32 Ncoeffs)
remainder            utilities.h     inline real4 remainder(real4 number, real4 divisor)
remainder            utilities.h     inline real8 remainder(real8 number, real8 divisor)
sinc                 utilities.h     inline real4 sinc(real4 x)
rect                 utilities.h     inline real4 rect(real4 x)
tri                  utilities.h     inline real4 tri(real4 x)
onedecimal           utilities.h     inline real8 onedecimal(real8 x)
onedecimal           utilities.h     inline real4 onedecimal(real4 x)
myrect               utilities.h     inline matrix<real4> myrect(const matrix<real4> &X)
myhamming            utilities.h     inline matrix<real4> myhamming(
interpbicubic        utilities.h     inline real4 interpbicubic(
normalize            utilities.h     inline real4 normalize(real4 data, real8 min, real8 max)
normalize            utilities.h     inline real8 normalize(real8 data, real8 min, real8 max)
```

Annex C

Utilities

In this annex the additional software that aids running Doris is described. This additional software is not required to run Doris, but it is highly recommended to use. We choose to only use freely available packages.

C.1 Packages

The program **getorb** for automatic retrieval of the Delft precise orbits for ERS1/2. See our web pages for a link.

The package gv (recommended) or ghostview to view postscript files, generated by **plotcpm** and **plotoffsets**, and **cpx2ps**.

The **GMT** (generic mapping tools). Highly recommended. We generally generate postscripts for visualizing the phase and amplitude with this program. See also **cpxfiddle** and **cpx2ps**.

C.2 Tools

We have developed some utilities for running Doris, and some display tools based on GMT.

C.2.1 Installation of SARtools

After downloading the gzipped, tarred archive of the SARtools, the installation is best done with a Makefile. I assume you are familiar with 'make' to compile code. If you are not, find someone who is. (See also annex B).

1. Use a distinct directory to put the files, e.g.,
   mkdir /opt/doris_v4.02
   cd /opt/Doris_v4.02
2. Download the archived SARtools via the download area of our webpages at http://enterprise.lr.tudelft.nl/doris/:
   SARtools.tar.gz
3. Expand compresed files: gzip -d SARtools.tar.gz
   This leaves a file **SARtools.tar**

4. Extract the files from the archive:
   tar -xvf SARtools.tar
   Now a subdirectory has been created, **SARtools**.

Now the compilation of the utilities can start.

1. Compile all utilities and install the executables:
   - cd SARtools (inspect/edit the Makefile, set INSTALLDIR)
   - make (this should compile the code)
   - make install (this installs in /usr/local/bin/)
2. Make sure the Installation directory is in your path. For tcsh users, it should be in your .cshrc file (startup file). Add it with a (csh) command like:
   `set path = ( /usr/local/bin/ $path )`

C.2.2 run script

This script can generate template **input files** and directories, and therefor generally speeds up the processing. If it is installed in a Bin directory (in your path), it can be run from several project directories for uniform processing. It uses the environment variables **EDITOR** and **PAGER** if set.

The basic idea is to start with

```
run -g
```

to **g**enerate the input, and then to to **e**dit (default with editor vi or EDITOR) the first **input file**.

```
run -e1
```

After you saved the file (located in directory Inputfiles), type:

```
run -s1
```

to process (call to doris software) the first **input file**. The output that goes to stdout can be **v**iewed with

```
run -v1
```

(It is redirected to a file in directoy Outinfo). To view the **r**esult files that are created by Doris, use

```
run -r1 (master);
run -r2 (slave);
run -r3 (products);
run -r4 (logfile)
```

This should be repeated for other steps. Of course this run file is only a helping hand, not the solution to all your problems. Be careful! Template values aren't always the best settings. Feel free to improve it.

```
Usage:
   run -s/-e/-v step  [-f \inputfile  -r file-id  -d]
         /-g [-M master  -S slave  -B baseline  -R remark  -A author]
         /-h
```

For more help type **run -h**.

C.2.3 cpxfiddle

With this utility one can fiddle about with binary complex (cpx) files of all kinds of formats (though only pixel interleaved). Make a cutout, multilook, scale, exponent, subsample, mirror, etc. Now it also supports the

generation of SUNraster files for visualization of the phase of complex files (smaller temp files required). It is written in C++ using a template function.

Input is a complex file, for example the output of Doris, or SLC data. It should be pixel interleaved, i.e., RE,IM, RE,IM, RE,IM, ... (Almost) all binary pixel interleaved formats are supported.

Output is written to stdout channel (normally your screen) in ascii or binary float format. The binary output should be redirected to a file or piped to a (GMT) program. Ascii output can best be used to view a small cutout. Output option (-o) are *normal* (the file "as is"), the *mag*nitude, the *phase*, the *real*, or the *imaginary* part. Further options are making a cutout, multilooking, subsampling, and/or mirroring in the vertical and horizontal plane.

Cpxfiddle does **not** handle band interleaved complex data, column major order files, nor non-complex files. However cpxfiddle might be tricked.

See also cpxfiddle -h and the utility cpx2ps (C.2.4).

```
SYNOPSIS:
      ./cpxfiddle -w width [-f informat] [-q output] [-o outformat]
      [-e exp] [-s scale] [-l line] [-L line] [-p pixel] [-P pixel]
      [-S x/y] [-M x/y] [-m mirror] [-c file] [-r rmin/rmax] [-B swap]
      [-H bytes] [-V] [-b] [-h[elp]] [--] [--ignorenan] [--fullstat] inputfile

 Dump content of complex binary file to stdout,
  either: as is, magnitude, phase, real or imaginary part.
 Input files can be almost any complex file
  though not (yet) band interleaved.
 Output can be manipulated by:
 multilooking, subsampling, mirroring, scaling, etc.
 This program is useful for cropping and displaying, in combination
 with e.g., GMT, ImageMagick, or xv.
 Output format to stdout can be binary.
 Careful! only pipe or redirect this.
 (use: "cpxfiddle -h |& more"     in csh
    or "cpxfiddle -h 2>&1 | less" in bash for more help.)
```

C.2.4 cpx2ps

With this utility postscript files from complex data (and binary float data) can be generated, such as the output of Doris and SLC files.

Various input formats are supported. Options are multilooking, mirroring, plotting the phase, magnitude, real, and imag part, etc. It has become pretty big.

It calls/requires **cpxfiddle** (see C.2.3), and **GMT** subprograms: grd2cpt, grdimage, psscale.

```
cpx2ps v2.1, FMR software, Bert Kampes, (c)1999-2000

PROGRAM:
  cpx2ps -- produce various encapsulated postscript code from
            complex data files.

SYNOPSIS:
  cpx2ps -w width [-f format==cr4] [-q out==mag] [-e exp==1.0] [-s sc==1.0]
       [-l 1] [-L alllines] [-p 1] [-P width] [-M1/1 | -F1/1]
       [-T title] [-c cptname==gray] [-z size==16]
       [-o epsfile] [-G grdfile] [-C cptfile]
       [-gKSUVZ] [-h elp] [--] cpxfile

[...]
```

For more info, type: cpx2ps -h | more

C.2.5 phasefilt.doris

Program to perform phase filtering from the prompt using Doris. Several methods can be used. For more info type: phasefilt.doris -h

```
PROGRAM:  phasefilt.doris  filter mph file using Doris.

SYNOPSIS:
  phasefilt.doris -l numlines [-o .filtered] [-a 0.25] [-s 2]
             [-e 3] [-b 32] [-m goldstein] [-k file] [--] infile
[...]
```

C.2.6 flapjack

Program to make integer linear combinations of interferograms.

```
PROGRAM: flapjack pixelwise complex integer multiplication of a
float complex file.  To be used to make linear combinations, ...

SEE ALSO: cpxmult...

USAGE: flapjack infile1 [factor==2]

EXAMPLE: flapjack cint.raw 3
```

C.2.7 cpxmult

Program to subtract phase in 2 complex files.

```
PROGRAM: cpxmult subtracts or adds phase of two complex float files

USAGE: cpxmult infile1 infile2 [outfile [add==0]]

infile[12] contain complex values a+ib
outfile contains (a1+ib1)*conj(a2+ib2)

If add=1 then phase is added (not conj).

EXAMPLE: cpxmult cint.raw cint2.raw subtract.raw
```

C.2.8 cpxdiv

Program to performs division on 2 complex files. Phase is subtracted by default.

```
Program: ./cpxdiv divides two given complex float files

 USAGE:
       ./cpxdiv infile1 infile2 [outfile [cnj==0]]

 infile[12] contain complex values a+ib
 outfile contains (a1+ib1)/(a2+ib2)

 If cnj=1 then phases are added.
 outfile contains (a1+ib1)/conj(a2+ib2)

 EXAMPLE: cpxdiv cint.raw cint2.raw division.raw
```

C.2.9 cpxconj

Program to take conjugate of a complex file.

```
Program: ./cpxconj take conjugate of a given complex float file

 USAGE:
       ./cpxconj  infile1  [ outfile ]

  infile[12] contain complex values a+ib
  outfile contains conj(a+ib)=a-ib

 EXAMPLE: cpxconj cint.raw cint.raw.conj
```

C.2.10 floatmult

This utility multiplies a (complex) float file by a scalar.

```
PROGRAM: floatmult pixelwise float multiplication of
a (complex) float complex file.
To be used to scale float files, or magnitude of complex files.

SEE ALSO: cpxmult, flapjack, cpxfiddle...

USAGE: floatmult infile1 [factor==2.]

EXAMPLE: floatmult cint.raw 1.47
```

C.2.11 wrap

With this utility you can wrap your interferogram to arbitrary interval instead of [-pi, pi). Can be used for example to make fringes correspond to, e.g., 1 cm displacement.

```
PROGRAM: wrap wraps float binary file to interval [a,b)

USAGE: wrap infile [a b [ofile]]

EXAMPLE: wrap interferogram.raw -4pi 4pi interf4.raw

default \outputfile    == infile.wrap
default interval [a b) == [-pi pi)
```

C.2.12 construct_dem.sh

This utility downloads, merges and fills voids of SRTM data based on coordinates of your area of interest. Only basic Linux/Unix commands and GMT are used (so make sure you have GMT installed).

```
PROGRAM: construct_dem.sh ownloads, merges and fills voids of SRTM data based on
coordinates of your area of interest.

USAGE: construct_dem.sh project W E S N SRTM[1|3] [<ftp1> <ftp2> <ftp_user> <ftp_pass>

EXAMPLE: construct_dem.sh netherlands 3.3 7.3 50.7 53.7 SRTM3

OUTPUT: - DEM: final_PROJECT.dem
        - preview: srtm_PROJECT.ps
        - Doris input: input.doris_PROJECT
```

C.2.13 doris.process-reset.sh

This utility can be used to reset and clean up the processing entries in Doris result files, such as: master.res, slave.res or interferogram.res, in order to allow re-processing of a step or multiple steps. Basicly, it deletes any entry starting from the indicated step till to the very end of the result file and updates process control switches.

```
USAGE: doris.process-reset.sh <doris_process_name> <.res>

   EX: doris.process-reset.sh coarse_correl   master_slave.res

       doris.process-reset.sh      resample   slave.res

Doris process names by order:
 1.  crop                13. resample
 2.  sim_amplitude       14. interfero
 3.  master_timing       15. comp_refphase
 4.  filt_azi            16. subtr_refphase
 5.  filt_range          17. comp_refdem
 6   oversample          18. subtr_refdem
 7.  coarse_orbits       19. coherence
 8.  coarse_correl       20. filtphase
 9.  fine_coreg          21. unwrap
10.  timing_error        22. slant2h
11.  dem_assist          23. geocoding
12.  comp_coregpm        24. dinsar
                         25. <extra>
```

C.3 Completes for tcsh users

Complete commands are used in tcsh shell to complete commands by pressing the TAB key. These completes can be added to the ones you already have. Simply put them in your resource file, likely .cshrc or .tcshrc. Possibly in your configuration there is a file ”complete.tcsh” that is sourced from the .cshrc. If you are not using tcsh, you cannot use them (?) (Find out by the commands ”who am i” and ”finger”.)

```
complete doris      c/-/"(c h q ver)"/ \
                    n/-h/"(<search term>)"/ \
                    n/*/f:*{in,IN,doris}*/

complete cpx2ps     c/-/"(w f q e s l L p P T F c z o G C g K S U V Z h m)"/ \
                    n/-f/"(ci2 cr4 cr8 r4)"/ \
                    n/-q/"(normal mag phase real imag)"/ \
                    n/-T/"(<title>)"/ \
                    n/-c/"(cool copper gebco gray haxby hot jet no_green polar
rainbow red2green relief topo sealand split wysiwyg)"/ \
                    n/-m/"(X XY Y)"/

complete cpxfiddle  c/-/"(w f q o e s l L p P S M m c V h)"/ \
                    n/-c/"(<filename> gray jet hot cool bert)"/ \
                    n/-f/"(cc1 cuc1 ci2 ci4 cr4 cr8)"/ \
                    n/-q/"(normal mag phase real imag)"/ \
                    n/-m/"(X XY Y)"/ \
                    n/-o/"(ascii float sunraster uchar)"/
```

Annex D

Definitions

In this annex a number of definitions as used by Doris are described. In Section D.2 the baseline representations are described, while in Section D.3 the definition of the interferogram is described. Section D.4 describes the definition of the polynomials. And in Section D.5 the use of the pulse repetition frequency and range sampling rate are discussed. Section D.6 describes a system of 3 equations which is frequently used to compute the position on the ellipsoid for a certain line, pixel. The way the orbits are interpolated is described in Section D.7. Finally section D.8 gives some information on the formats of the images.

D.1 Constants

Constants used in the processing can be found in the source files constants.h and refsystems.h. The main parameters in constants.h are:

```
const real8     SOL         = 299792458.;    // speed of light in m/s
const real8     EPS         = 1e-13;         // small number
const int32     NaN         = -999;          // Not a Number
const real8     PI          = 4.*atan(1.);
```

The main parameters in refsystems.h are (actually **only WGS** is used for now):

```
const   real8   WGS84_A=6378137.0;              // semimajor axis wgs84
const   real8   WGS84_B=6356752.3;              // semiminor axis wgs84
const   real8   GRS80_A=6378137.0;              // semimajor axis grs80
const   real8   GRS80_B=6356752.3;              // semiminor axis grs80
const   real8   BESSEL_A=6377397.155;           // semimajor axis bessel
const   real8   BESSEL_B=6356078.963;           // semiminor axis bessel
const   real8   RADIUS=.5*(WGS84_A+WGS84_B);    // for sphere pol2car
```

D.2 Baseline

The basic configuration of InSAR is shown in figure D.1.

There are different representations for the baseline, see Figure D.2.

Conversions between baseline representations:

The baseline parameters can be computed when the statevectors of the points M, S and P (master ,slave and point on surface) are known. (The distance between the points *x* and *y* is denoted by $\mathrm{d(x,y)}$; and the sharp

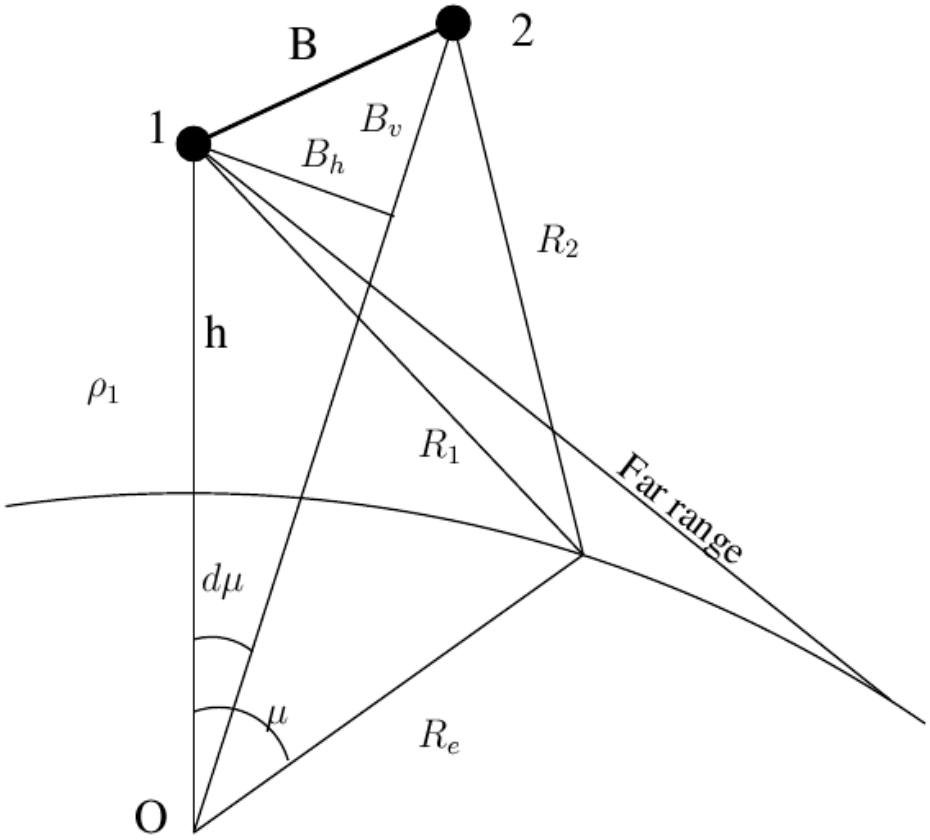

Figure D.1: Geometric configuration for InSAR. R_i are the range vectors to the corresponding resolution element. The statevector of the reference satellite is denoted by ρ_1. (h denotes the satellite height, μ the location angle of the co-registered resolution cell in the interferogram.)

angle between two vectors *x* and *y* by $\angle(x, y)$.)

$$B = \mathrm{d}(M, S) \tag{D.1}$$

$$B_{\parallel} = \mathrm{d}(M, P) - \mathrm{d}(S, P) \tag{D.2}$$

Now the perpendicular baseline has to be computed. The definition states that $B_{\perp}$ is positive if the slave satellite is to the right of the slant range line of the master. Which yields for a mountain an increasing phase from foot to summit(?) We had some trouble finding a simple expression to find out the correct sign but at the moment we do something like the following.

$${B_{\perp}}^2 = B^2 - {B_{\parallel}}^2 \tag{D.3}$$

$$\vec{r}_1 = \vec{M} - \vec{P} \tag{D.4}$$

$$\vec{r}_2 = \vec{S} - \vec{P} \tag{D.5}$$

$$\gamma_1 = \angle(\vec{P}, \vec{r}_1) \tag{D.6}$$

$$\gamma_2 = \angle(\vec{P}, \vec{r}_2) \tag{D.7}$$

$$\mathrm{sign} = \begin{cases} -1 & ,\gamma_1 < \gamma 2 \\ 1 & ,\gamma_1 > \gamma 2 \end{cases} \tag{D.8}$$

$$B_{\perp} = \mathrm{sign}\sqrt{{B_{\perp}}^2} \tag{D.9}$$

Table D.1: Conversion between baseline representations (note that the four quadrant arctangent should be used).

	$[B_h, B_v]$	$[B, \alpha]$	$[B_\perp, B_\parallel]$
$[B_h, B_v]$	–	$B_h = B\cos\alpha$	$B_h = B_\perp\cos\theta + B_\parallel\sin\theta$
	–	$B_v = B\sin\alpha$	$B_v = B_\perp\sin\theta - B_\parallel\cos\theta$
$[B, \alpha]$	$\alpha = \arctan(B_v/B_h)$	–	$\alpha = \theta - \arctan(B_\parallel/B_\perp)$
	$B = \sqrt{B_h^2 + B_v^2}$	–	$B = \sqrt{{B_\parallel}^2 + {B_\perp}^2}$
$[B_\perp, B_\parallel]$	$B_\parallel = B_h\sin\theta - B_v\cos\theta$	$B_\parallel = B\sin(\theta - \alpha)$	–
	$B_\perp = B_h\cos\theta + B_v\sin\theta$	$B_\perp = B\cos(\theta - \alpha)$	–

$$\theta = \angle(\vec{M}, \vec{r}_1) \tag{D.10}$$

$$\alpha = \theta - \arctan 2(B_\parallel, B_\perp) \tag{D.11}$$

$$B_h = B\cos(\alpha) \tag{D.12}$$

$$B_v = B\sin(\alpha) \tag{D.13}$$

D.3 Interferogram

The phase for a certain pixel in a single SLC image *i* is defined as:

$$\phi_i \equiv -\frac{4\pi}{\lambda} r_i \tag{D.14}$$

The complex interferogram minus reference phase is defined as:

$$I = M \cdot S^* \cdot R^* \tag{D.15}$$

Where:
$\{.\}^*$ denotes the complex conjugated;

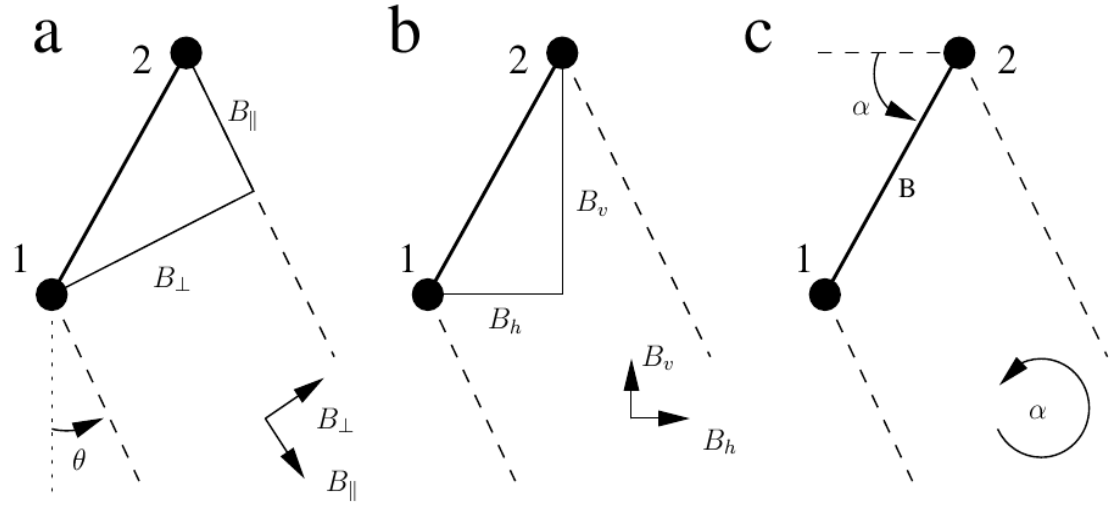

Figure D.2: Definition of the baseline parameters. (a) parallel/perpendicular; (b) horizontal/vertical; (c) length-/orientation; Position 1 is the reference position. $B_\parallel > 0$ when $R_1 > R_2$, where R_i is the corresponding slant range. The angle α is defined counter-clockwise from the reference satellite (1), starting from the horizontal at the side of the look direction.

$\cdot$ denotes a pointwise multiplication;
I is the complex interferogram;
M is the complex master image;
S is the complex (resampled) slave image;
R is the complex (amplitude $\equiv 1$) reference phase;
The phase image (of complex interferogram minus reference phase) is defined as:

$$\phi = \arctan_2(I_{\text{imag}}, I_{\text{real}}) \tag{D.16}$$

Where:
$\arctan_2$ is the four quadrant arc tangent;
ϕ is the phase image;
I is the complex interferogram;
Which is equal to (with an ambiguity of 2π)

$$\phi_I = \phi_M - \phi_S - \phi_R \tag{D.17}$$

The reference phase is defined as (where r_1 denotes the range from the master satellite to a point on the reference surface)

$$\phi_R \equiv -\frac{4\pi}{\lambda}(r_1 - r_2) = -\frac{4\pi}{\lambda}B_{\|} \tag{D.18}$$

Which is the same as

$$R = M_r \cdot S_r^* \tag{D.19}$$

Where M_r denotes the phase of a point situated on the reference surface. (of course, in this definition the phase of the interferogram equals zero if there actually is no topography (and $M = M_r$).) The values of the (real valued) reference phase are stored in a 2d-polynomial of certain degree. The subtraction of the reference phase ϕ is actually computed as:

$$I = (M \cdot S^*) \cdot (\cos\phi - \sin\phi) \tag{D.20}$$

because the complex conjugated of the reference phase ϕ equals:

$$(ae^{i\phi})^* = a(\cos\phi + \sin\phi)^* \overset{a\equiv 1}{=} \cos\phi - \sin\phi \tag{D.21}$$

The complex coherence between two images is defined as (see [Touzi et al., 1996]):

$$\gamma_c = \frac{E\{M \cdot S^*\}}{\sqrt{E\{M \cdot M^*\} \cdot E\{S \cdot S^*\}}} \tag{D.22}$$

Where:
$E\{.\}$ is the expectation;
* is the complex conjugated;
γ_c is the complex coherence;
M is the complex master image;
S is the complex slave image (possibly minus (complex) reference phase: $S = S \cdot R^*$;
The coherence is defined by $|\gamma_c|$, and its estimator as:

$$\hat{\gamma} = \left| \frac{\frac{1}{N}\sum_{i=0}^{N} M_i S_i^*}{\sqrt{\frac{1}{N}\sum_{i=0}^{N} M_i M_i^* \frac{1}{N}\sum_{i=0}^{N} S_i S_i^*}} \right| \tag{D.23}$$

The correlation between two images is defined by (see [Bähr and Vögtle, 1991]):

$$\Gamma = \frac{\text{cov}(M,S)}{\sqrt{\text{var}(M)\text{var}(S)}} = \frac{E\{M \cdot S^*\} - E\{M\}E\{S^*\}}{\sqrt{(E\{M \cdot M^*\} - E\{M\}E\{M^*\}) \cdot (E\{S \cdot S^*\} - E\{S\}E\{S^*\})}} \tag{D.24}$$

Thus the mean is first subtracted in comparison to the coherence. The coherence is equal to the correlation only if $E\{M\} = E\{S\} = 0$.

A problem is that the estimator for the coherence and correlation is biased. For small window sizes its outcome is too high. This probably also causes the problems in the coarse coregistration, where the most likely offset is not selected based on its correlation value but on its consistency.

D.4 Polynomials

A 1d-polynomial is defined as:

$$f(x) = \sum_{i=0}^{d} \alpha_i x^i \tag{D.25}$$

A 2d-polynomial is defined as:

$$f(x,y) = \sum_{i=0}^{d} \sum_{j=0}^{i} \alpha_{i-j,j} x^{i-j} y^j \tag{D.26}$$

Thus the order of the coefficients (line,pixel) is independent of degree d:
d=0: A_{00} (1)
d=1: $A_{10}A_{01}$ (2, 3)
d=2: $A_{20}A_{11}A_{02}$ (4, 5, 6)
d=3: $A_{30}A_{21}A_{12}A_{03}$ (7, 8, 9, 10)

Thus the number of coefficients (unknowns in least squares estimation) equals for a 2d-polynomial of degree d:

$$\frac{1}{2}((d+1)^2 + d + 1) \tag{D.27}$$

And the degree of a polynomial with *N* coefficients is equal to:

$$d = \frac{1}{2}(int32(\sqrt{1+8N}) - 1) - 1 \tag{D.28}$$

D.4.1 Computation of coefficients

Suppose we have 2d data f(l,p), l[1,25000] p[1,5000] and we want to estimate a 2d polynomial D.26 with these data. The system of equations looks like

$$\begin{bmatrix} f(l_1,p_1) \\ f(l_2,p_2) \\ \vdots \\ f(l_N,p_N) \end{bmatrix} = \begin{bmatrix} 1 & l_1 & p_1 & l_1^2 & l_1p_1 & \cdots & p_1^d \\ 1 & l_2 & p_2 & l_2^2 & l_2p_2 & \cdots & p_2^d \\ \vdots & & & & & & \\ 1 & l_N & p_N & l_N^2 & l_Np_N & \cdots & p_N^d \end{bmatrix} \begin{bmatrix} \alpha_{00} \\ \alpha_{10} \\ \alpha_{01} \\ \alpha_{20} \\ \alpha_{11} \\ \vdots \\ \alpha_{0d} \end{bmatrix} \tag{D.29}$$

The convention used in the Doris software is that we first normalize the data to avoid numerical instabilities (see source **utilities.[hc]**). The maximum coordinates are that of the original master (stored in the **result file** of the master image, typically 25000 lines and 5000 range pixels). The coordinates are rescaled to the interval [-2,2].

$$l[a,b] \rightarrow l[-2,2] \Leftrightarrow l \rightarrow 2\frac{l-a}{.25(b-a)} - 2 \tag{D.30}$$

(Another way, perhaps a better one (?), would be to make the data zeromean, unit standard deviation) The estimated coefficients thus correspond to the normalized data. For evaluation, the data has to be normalized by the same factors a,b. Normally the information from the master.originalwindow.linelo etc. are used, e.g., for the coregistration and the reference surface polynomial. These numbers can be found in the **master result file** after the step readfiles at place number of lines original of datafile. A function *normalize* is called to do the normalization, so it is easy to change the implementation to a different normalization. It has been noticed that for higher order polynomials the normalization factor is very important to obtain a stable estimate.

D.4.2 Evaluation of polynomials

Evaluation of the polynomials should be done by normalizing the data as indicated above. Something like:

```
const real8 const real8 minL = master.originalwindow.linelo;
const real8 maxL = master.originalwindow.linehi;
const real8 minL = master.originalwindow.pixlo;
const real8 maxL = master.originalwindow.pixhi;
matrixreal4(N,1) l\_axis = linenumbers;
normalize(l\_axis);
matrixreal4 f = polyval(l\_axis, p\_axis, coefficients, [degree]);
```

NOTE: It is faster to evaluate a polynomial on a grid than point by point.

D.5 (SAR) System parameters

D.5.1 Azimuth

PRF

The actual pulse repetition frequency (PRF) is computed based on the data in the SLC leader file. However, the 'actual' value, as read from the leader file, is used (after private communications with ESA helpdesk). It is defined as:

$$\mathrm{PRF} = \frac{Nl - 1}{dt_a} \tag{D.31}$$

Where:
PRF is the pulse repetition frequency in Hz.
Nl is the total number of lines (lastline - firstline).
dt_a is the azimuth time of the last line minus the azimuth time of the first line, or the acquisition time of the image.
This equation, and the following, can be easily verified by substitution of the values for the first/last line/pixel.

line number

The azimuth time of a certain line (number) t_{al} is computed as:

$$t_{al} = t_{a1} + \frac{(l-1)}{\mathrm{PRF}} \tag{D.32}$$

Where:
t_{a1} is the azimuth time to line 1 (first line).

And the line number *l*, given a certain azimuth time t_a, can be computed as:

$$l = 1 + \mathrm{PRF}(t_a - t_{a1}) \tag{D.33}$$

Where:
t_{a1} is the azimuth time to line 1 (first line).

Doppler centroid

The Doppler centroid frequency (azimuth) is computed as a second degree polynomial:

$$f_{DC} = \alpha_0 + \alpha_1 \frac{p}{RSR} + \alpha_2 \left[\frac{p}{RSR}\right]^2 \tag{D.34}$$

Where:
p is the pixel number starting at 0.
α_i is read from the leader file.
In the **master result file** it are the variables (e.g.):

```
Xtrack_f_DC_constant (Hz, early edge):              117.3210000
Xtrack_f_DC_linear (Hz/s, early edge):            72338.0000000
Xtrack_f_DC_quadratic (Hz/s/s, early edge):    −455000000.00000
```

This frequency is used in the azimuth filtering, and in the resampling. It should also be used if the complex SLC data is harmonically oversampled (as is done in the range filtering routine), but we did not implement this yet. But, for $\| f_{DC} \| < 150$ Hz this should not have any effect (assuming PRF-ABW=300Hz).

Since for a signal f(t) with Fourier transform F(ω)

$$f(t) \quad \overset{FT}{\longleftrightarrow} F(\omega) \tag{D.35}$$

$$e^{j\omega_0 t} f(t) \quad \overset{FT}{\longleftrightarrow} F(\omega - \omega_0) \tag{D.36}$$

(D.37)

The azimuth spectrum of a SLC image processed on a certain Doppler frequency f_{DC} (spectrum shifted to this frequency) can be shifted back to zero by multiplication in the space domain by the term

$$e^{-j2\pi \frac{f_{DC}}{PRF} line} \tag{D.38}$$

(See also any signals and systems book, or e.g., [Geudtner, 1996].) The spectrum can be shifted back to the original doppler centroid frequency by multiplication by (after e.g., interpolation):

$$e^{j2\pi \frac{f_{DC}}{PRF} line'} \tag{D.39}$$

Proper care should be taken to get the correct line number in both situations.

D.5.2 Range

RSR

The range sampling rate (RSR) is defined as:

$$\mathrm{RSR} = 0.001 \frac{Np - 1}{dt_r} \tag{D.40}$$

Where:
RSR is the range sampling rate in MHz.
Np is the number of (range) pixels.
dt_r is the zero Doppler **two-way** time to the last pixel minus the range time to the first pixel in milliseconds.

pixel number

The pixel number *p* [1:Np], given a certain **one-way** range time t_a, can be computed as:

$$p = 1 + \mathrm{RSR} \cdot 2(t_r - t_{r1}) \tag{D.41}$$

Where t_{r1} is also one-way.

The one-way range time for a given pixel (number) *p* can be computed as:

$$t_r = t_{r1} + \frac{(p-1)}{2\mathrm{RSR}} \tag{D.42}$$

Where:
RSR is in Hz.
t_{a1} is the range time to pixel 1 (first pixel) in seconds.

The range is of course equal to:

$$r = t_r * c \tag{D.43}$$

Where:
c is the speed of light (constants.h: 299792458. m/s).

D.6 Doppler, range and ellipsoid equations

The following three equations are used regularly throughout the software to compute the point P that corresponds to a certain line and pixel in the master or slave image (see also [Geudtner, 1996]). Precise orbits are necessary.

1. Doppler: The point P at the surface lies perpendicular to the orbit due to zero Doppler processing (otherwise this equation has to be adapted with a slant angle).
2. Range: The geometrical distance to P on the surface is equal to the speed of light times the range time.
3. Ellipsoid: Force the point to lie on an ellipsoid.

The equations for the point P on the ellipsoid and the satellite S in its orbit are (where x denotes (x,y,z)):

$$dx = x - x_s \tag{D.44}$$

$$E_1 : \dot{x_s} \cdot dx = 0 \tag{D.45}$$

$$E_2 : dx \cdot dx - (v_{\mathrm{light}} t_{\mathrm{range}})^2 = 0 \tag{D.46}$$

$$E_3 : \frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{b^2} - 1 = 0 \tag{D.47}$$

To compute the coordinates of a point P on the ellipsoid, corresponding with line *l* and pixel *p* in the master image the following has to be done. First the position of the satellite has to be computed (assumed exact) based on the line number and PRF (l to azimuth time to interpolated position), and the velocities for this time (by interpolation). Also the range time corresponding to the pixel number is computed (based on RSR, assumed exact).

Next the set of equations is used to solve for P(x,y,z). This is done iteratively by linearization, which requires the derivative of the equations to x and approximate values for the unknowns (the coordinates of the center (ϕ, λ) given in the SLC leader file, converted to xyz on a sphere).

$$dx = x - x_s \tag{D.48}$$

$$\begin{bmatrix} \frac{\delta E_1}{\delta x} & \frac{\delta E_1}{\delta y} & \frac{\delta E_1}{\delta z} \\ \frac{\delta E_2}{\delta x} & \frac{\delta E_2}{\delta y} & \frac{\delta E_2}{\delta z} \\ \frac{\delta E_3}{\delta x} & \frac{\delta E_3}{\delta y} & \frac{\delta E_3}{\delta z} \end{bmatrix} = \begin{bmatrix} \dot{x_s} & \dot{y_s} & \dot{z_s} \\ 2\,dx & 2\,dy & 2\,dz \\ \frac{2x}{a^2} & \frac{2y}{a^2} & \frac{2z}{b^2} \end{bmatrix} \tag{D.49}$$

Solving this exactly determined system of 3 equations yields the next solution dx_1 and the new values for the unknowns become $dx_1 = dx_0 + dx_1$ which are used to compute dy_1 and dA_1. The solution is updated until convergence ($\Delta x <$ 1e-6 meters).

$$dy_i = dA_i\, dx_i \tag{D.50}$$

Where:
dy contains the observations. (set of equations)
dx contains the unknowns (coordinates of P). dA contains the partials (evaluated for previous solution).

To solve for the azimuth time if the coordinates of a point on the ground is known, only the Doppler D.45 equation needs to be used. the derivative with respect to azimuth time of this equation equals

$$\frac{\delta E_1}{\delta t_a} = \ddot{x}_s \cdot dx - \dot{x}\dot{x} \tag{D.51}$$

The solution is equal to (use approximate solution t_{a0} to evaluate these expressions).

$$t_{a1} = \frac{-E_1}{\frac{\delta E_1}{\delta t_a}} \tag{D.52}$$

and

$$t_{a1} = t_{a0} + t_{a1} \tag{D.53}$$

The solution is updated until convergence ($\Delta t <$ 1e-10 seconds).

The range time is then computed as in equation D.46

$$t_r = \frac{\sqrt{(x - x_s)^2}}{c} \tag{D.54}$$

D.7 Orbit interpolation

We assume the precise orbits are given some time before the first and after the last azimuth line. Normally we use getorb to obtain satellite ephemerides with a time interval of 1 second (approximately 21 datapoints for a frame of typical 15 seconds).

Natural cubic splines are then used to interpolate the orbit. Because these splines do not behave very well at the edges we use some points before/after the first/last line. Note that the x, y, and z coordinate are interpolated independently.

The Delft precise orbits and the getorb package are used to obtain the points. Note that getorb also interpolates based on 30 second ephemerides.

We would like to test if setting the data interval to e.g. 30 second gives better results. A test can be easily performed for the computation and modeling of the reference phase. Assume that this phase can be accurately modeled by a 2d polynomial of degree 5. Now first let the precise orbit be given with a data interval of 1 second. Use step REFPHA to model the reference phase based on 501 points distributed over the scene. Next, let the precise orbit be given with a data interval of 30 seconds and again model the reference phase. In the log file some statistics on the error of the model w.r.t. the computed reference phase is given, which can be used to find out which orbit gives a better model. In both cases use at least 6 points before and 6 points after the last point in the frame.

The interpolation is done as follows, compare with numerical recipes in c (splint routine). First the piecewise polynomial coefficients are computed by solving a tridiagonal system and stored. For interpolation, the correct coefficients (interval) are read and the polynomial is evaluated.

Because we know that in our situation (with getorb) we always have ephemerides with a constant time interval we could speed up the computations. Also the fact that this interval equals 1 can be easily exploited. However, we decided not to exploit these features because we like to stay independent from a particular orbit format. (and these computations can be done fast anyhow.)

The velocity can be interpolated by the derivative of the piecewise polynomials: [see source code or numerical recipes].

The accerelation can be interpolated by the second derivative of the piecewise polynomials: [see source code or numerical recipes].

If less points are known (than the typically 21 of getorb) (one wants to use the SLC datapoints for a quick look analysis for example) then this kind of interpolation probably does not work very well. In future we will include an option to interpolate by a low degree polynomial which is estimated (least squares) from the datapoints. Getting the derivates at any point is straightforward in this case.

As a satellite moves very smoothly, a polynomial of a lower degree might even be nearer to the 'true' orbit then a piecewice polynomial. In future we want to model the baseline (Bh, Bv) as a function of azimuth time by a first? order polynomial. This probably is more efficient than computing the positions of the sensors each time the baseline is required. We do not know what the best way is to do this.

D.8 Format of the products

Start at (azimuth) line 1, (range) pixel 1 (near range). Data is written line by line (major row order). We give the binary data a .raw extension.

The complex interferogram is written pixel interleaved (see D.2). Each complex pixel is written as 4 byte real, 4B imaginary part.

Table D.2: The way complex files (SLC data, resampled slave, complex interferogram) are stored on disk. Also refered to as mph format (magnitude phase). This is a major row order stored, pixel interleaved file with 2 (float 4B) canals (real,imag).

	1st pixel	2nd pixel	...	pixel P
1st line	Real,Imag	Real,Imag	...	Real,Imag
2nd line	Real,Imag	Real,Imag		...
3rd line	Real,Imag	Real,Imag		
...	...			
line L	Real,Imag	Real,Imag	...	Real,Imag

After unwrapping of the phase the result can no longer be stored as a complex value, because a complex number only can distinguish between phase values in the principal interval $\pm\pi$. Therefor a new format is used/will be used. Either the unwrapped phase is simply stored in a 4B float file, similar to table D.2 without the imaginary part (as are other files, like the phi, lambda, and height matrices after geocoding), or a hgt file is generated (see table D.3.

Particularly after unwrapping the conventions we use for this file are as follows. The amplitude (equal to that of complex intererogram) and the unwrapped phase is stored for each pixel. If there is no unwrapped phase, the wrapped phase is stored, and the amplitude is set to 0 for that pixel. (I would prefer setting the phase to 0, and keeping the amplitude, but we selected this format to keep in line with other software.) The amplitude is stored, while it does not change after unwrapping, to keep all information in one file.

Table D.3: Format of a hgt (height) file. (unwrapped complex interferogram and others) Actually is a major row order, band interleaved data, with 2 float (4B) canals (amplitude phase).

	1st pixel	2nd pixel	...	1st pixel	2nd pixel	...
1st line	Amplitude	Amplitude	...	Phase	Phase	...
2nd line	Amplitude	Amplitude		Phase	Phase	...
3rd line	Amplitude	Amplitude		...		
...	...					
line L	Amplitude	Amplitude	...	Phase	Phase	...

Computations are done in general in the original master system. (no matter if cut-out or multilooked.)

Time system (orbit) is in seconds of day. Ephemerides orbit system is more or less WGS84.

Matrix class:
A matrix is starts at 0,0 etc.

Offset:
Offset for a certain point is defined as: coordinate in slave system = coordinate in mastr system + offsets.

Annex E

Matrix class

The template matrix class (called **matrixbk**) that is provided with the Doris software can be used for other applications as well. Please refer to the file matrixbk_test.c for an example how to use this class in your own programs. (I do not claim it is the best/fastest implementation ever, i just find it very useful, and the routines are checked and working fine.) The Makefile shows how to compile it. Also see Annex B.

The data has to be linear in memory for the VECLIB library, thus this has been done and it used in some other functions to speed them up (see the constructor how this is done).

If the VECLIB library is not used, (slower) internal functions for FFT and matrix multiplication are used. If you like to use your own FFT, only the function four1d has to be changed since the 2d function call this one sequentially.

If the LAPACK library is not used, (slower) internal functions for cholesky are used.

The matrix class is used as a container class for (part of) the images. It has not been defined as a class 'radarimage' because in that case it would have been difficult to perform operations on the images if they didn't fit in the memory as a whole. However, it might be a good idea to define functions such as phasefilter for the matrix class. This would result in calls like:

```
matrix < complex<real4> > BUFFERMASTER;    // Container
for (i=0; i<NUMBUFFERS; ++i)
  {
    // Read in phase image
    BUFFERMASTER.readfromfile(filename,windowsize,formatflag);
    // filter this buffer
    BUFFERMASTER.phasefilter(parameters);
    // Write to \outputfile
    BUFFERMASTER.writetofile(filename2,formatflag);
  }
```

which seems very readable and maintainable (only the member function "phasefilter" of the matrix class has to be changed if something has to be added).

If you do not have access to LAPACK, but you have a different library, we advice you to use that one. The Cholesky factorization as implemented internally has not been optimized in any way. The same holds for the VECLIB library, particularly the FFT routines. We have included the code for a Cooley-Turkey algorithm, but it handles the data quite slowly, compared to an optimized library.

E.1 Matrix class functions

The functions in the matrix class are obtained by:

```
~/.bin/ctags --c-types=f -x matrixbk.* | cut -c1-12,50-600
```

```
0  allocate          void matrix<Type>::allocate(uint numlines, uint numpixels) // allocator
1  checkindex        void matrix<Type>::checkindex(uint line, uint pixel) const
2  clean             void matrix<Type>::clean() // sets 2 zero
3  conj              void matrix<Type>::conj()
4  conj              matrix<Type> conj (const matrix<Type> &A)
5  correlate         matrix<real4> correlate (const matrix<Type> &A, matrix<Type> Mask)
6  diagxmat          matrix<Type> diagxmat (const matrix<Type> &diag, const matrix<Type> &B)
7  dotdiv            matrix<Type> dotdiv (const matrix<Type> &A, const matrix<Type> &B)
8  dotmult           matrix<Type> dotmult (const matrix<Type> &A, const matrix<Type> &B)
9  dumpasc           void dumpasc(const char *file, const matrix<Type>& A)
10 fftshift          friend void fftshift (matrix<Type> &A)
11 fliplr            void matrix<Type>::fliplr()
12 flipud            void matrix<Type>::flipud()
13 getcolumn         matrix<Type> matrix<Type>::getcolumn(uint pixel) const
14 getdata           matrix<Type> matrix<Type>::getdata(window win) const
15 getrow            matrix<Type> matrix<Type>::getrow(uint line) const
16 ifftshift         friend void ifftshift (matrix<Type> &A)
17 initialize        void matrix<Type>::initialize(uint numlines, uint numpixels)
18 isvector          bool matrix<Type>::isvector() const
19 lines             uint matrix<Type>::lines() const // return number of lines
20 matTxmat          matrix<Type> matTxmat(const matrix<Type> &A, const matrix<Type> &B)
21 matrix            matrix<Type>::matrix() // constructor (0 arg)
22 matrix            matrix<Type>::matrix(uint lines, uint pixels)
23 matrix            matrix<Type>::matrix(const matrix<Type>& A)
24 matrix            matrix<Type>::matrix (window win, const matrix<Type>& A)
25 matxmatT          matrix<Type> matxmatT(const matrix<Type> &A, const matrix<Type> &B)
26 max               Type max(const matrix<Type> &A)
27 max               Type max(const matrix<Type> &A, uint& line, uint& pixel)
28 mean              real8 mean (const matrix<Type> &A)
29 min               Type min(const matrix<Type> &A)
30 min               Type min(const matrix<Type> &A, uint& line, uint& pixel)
31 multilook         matrix<Type> multilook (const matrix<Type> &A, uint factorL, uint factorP)
32 mypow             void matrix<Type>::mypow(Type s)
33 myswap            friend void myswap (matrix<Type> &A, matrix<Type> &B)
34 operator !=       bool matrix<Type>::operator != (Type scalar) const
35 operator !=       bool matrix<Type>::operator != (const matrix<Type> &A) const
36 operator *        matrix<Type> operator * (const matrix<Type>& A, const matrix<Type>& B)
37 operator *        matrix<Type> operator * (const matrix<Type>& A, Type scalar)
38 operator *        matrix<Type> operator * (Type scalar, const matrix<Type> &A)
39 operator *=       matrix<Type>& matrix<Type>::operator *= (Type scalar)
40 operator *=       matrix<Type>& matrix<Type>::operator *= (const matrix<Type> &A)
41 operator +        matrix<Type> operator + (const matrix<Type>& A, const matrix<Type>& B)
42 operator +        matrix<Type> operator + (const matrix<Type>& A, Type scalar)
43 operator +=       matrix<Type>& matrix<Type>::operator += (const matrix<Type>& A)
44 operator +=       matrix<Type>& matrix<Type>::operator += (Type scalar)
45 operator -        matrix<Type> operator - (const matrix<Type>& A, const matrix<Type>& B)
46 operator -        matrix<Type> operator - (const matrix<Type>& A, Type scalar)
47 operator -        matrix<Type> operator - (const matrix<Type>& A)
48 operator -=       matrix<Type>& matrix<Type>::operator -= (const matrix<Type>& A)
49 operator -=       matrix<Type>& matrix<Type>::operator -= (Type scalar)
50 operator /        matrix<Type> operator / (const matrix<Type>& A, Type scalar)
51 operator /        matrix<Type> operator / (const matrix<Type>& A, const matrix<Type>& B)
52 operator /=       matrix<Type>& matrix<Type>::operator /= (Type scalar)
53 operator /=       matrix<Type>& matrix<Type>::operator /= (const matrix<Type> &A)
54 operator <<       friend ostream& operator << (ostream& file, const matrix<Type>& A)
55 operator =        matrix<Type>& matrix<Type>::operator = (const matrix<Type>& A)
56 operator ==       bool matrix<Type>::operator == (Type scalar) const
57 operator ==       bool matrix<Type>::operator == (const matrix<Type> &A) const
58 operator >>       friend istream& operator >> (istream& file, matrix<Type>& A)
59 operator [        Type* matrix<Type>::operator [] (uint line) const
60 operator()        Type& matrix<Type>::operator () (uint line, uint pixel) const
61 operator()        matrix<Type> matrix<Type>::operator () (window win) const
```

```
operator()      matrix<Type> matrix<Type>::operator () (uint l0,uint lN,uint p0,uint pN) const
pixels          uint matrix<Type>::pixels() const // return number of pixels
readfile        friend void readfile(matrix<Type> &Result, const char *file,
resize          void matrix<Type>::resize(uint l1, uint p1)
setcolumn       void matrix<Type>::setcolumn(uint pixel, const matrix<Type> &COLUMN)
setcolumn       void matrix<Type>::setcolumn(uint pixel, Type scalar)
setdata         void matrix<Type>::setdata(Type w) // sets 2 w
setdata         void matrix<Type>::setdata(uint l1, uint p1, const matrix<Type>& A)
setdata         void matrix<Type>::setdata(window winin, const matrix<Type> &A, window winA)
setdata         void matrix<Type>::setdata(const matrix<Type> &A, window winA)
setrow          void matrix<Type>::setrow(uint line, const matrix<Type> &LINE)
setrow          void matrix<Type>::setrow(uint line, Type scalar)
showdata        void matrix<Type>::showdata() const // show all data in matrix
size            uint matrix<Type>::size() const // return nsize
sqr             matrix<Type> sqr (const matrix<Type> &A)
sqrt            friend matrix<Type> sqrt (const matrix<Type> &A)
sum             matrix<Type> sum (const matrix<Type> &A, int32 dim)
writefile       friend void writefile (
wshift          friend void wshift (matrix<Type> &A, int32 n)
~matrix         matrix<Type>::~matrix()
```

Annex F

Adding a module

In this annex a description is given how to add a module to the Doris software.

First get general idea of the structure of the software. (source2html?).

It is preferred to stay in same format as us.

General:

1. Read input cards and parameters for your module.
2. Add your module to the big selecting switch in main.
3. Implement your module, let in **result file** the output section end with same string as the other modules (END..._NORMAL).
4. Documentation (author date description, for users and code developers)
5. Email the description and the total source to the owner of the mailinglist *doris_users@tudelft.nl*. And if approved, we will include your functions in the next version of Doris.

F.1 Formats

The example source code explains which rules for commenting I generally follow.

```
0  /*******************************************************
1   *    ts16                                             *
2   *                                                     *
3   * truncated sinc 16 points                            *
4   *                                                     *
5   * input:                                              *
6   *  - x-axis                                           *
7   * output:                                             *
8   *  - y=f(x); function evaluated at x                  *
9   *                                                     *
10  *    Bert Kampes, 16-Mar-1999                         *
11  *******************************************************/
12  matrix<real4> ts16(
13          const matrix<real4> &x)
14    {
15  #ifdef _DEBUG
16    DEBUG("ts16.");
17    if (x.pixels() != 1)
```

```
18      ERROR("ts16: standing vectors only.");
19  #endif
20    matrix<real4> y(x.lines(),1);
21    for (register int32 i=0;i<y.lines();i++)
22      y(i,0) = sinc(x(i,0)) * rect(x(i,0)/16.);
23    return y;
24    } // END ts16
```

- start routine a block with date/author/description/input/output;
- end routine met // END routinename;
- no block comments inside the function, only things like:
  // ====== Comment on block ======
  // ____ Comment on something smaller ____

Indenting is done with 2 spaces, with the curly braces as shown below.

```
if (expression)
  {
    action1;
    action2;
  }
```

You can display information (depending on the value of the variable displevel) with the functions:

```
DEBUG(char[ONE27]);
INFO(char[ONE27]);
PROGRESS(char[ONE27]);
WARNING(char[ONE27]);
ERROR(char[ONE27]);
```

F.2 Adding a Step

Adding a new step is not intended to be necessary. The only thing that needs to be added are modules (methods) in pre-defined steps. However we will explain what you will have to change if you want to add a new step.

In file readinput.h:

1. you will have to add a const for the new step which is later stored in the process array;
2. also a struct has to be made to store the variables of this new step. (method selector, output file name, window sizes, etc.);
3. the prototype of the function readinput should be augmented with this new struct.

In the file readinput.c:

1. function readinput augment with new struct;
2. (only)process card, define new keyword for this step;
3. add reading of parameters into defined inputstruct by new keywords.

In file ioroutines.c: (only minor adding)

1. routine: doinitwrite: add new step
2. routine: initwrite: process control
3. routine: updateprocesscontrol: check for string
4. routine: checkprocesscontrol: check for string
5. routine: fillcheckprocesscontrol: check for string
6. routine: fillprocessed: check for string

In file processor.c:

1. add definition of new struct,
2. readinput augment with new struct,
3. add in big switch what to do if new step is requested.

General:

- document what you did, new keywords and arguments, new **process control flag**.
- how does the **result file** has to end? ”* END_filtphase:_NORMAL”
- what strings in the **result file** are used later in the program?
- email to owner-doris_users@tudelft.nl.